

Validating the Grid Diversity Operator: An Infusion Technique for Diversity Maintenance in Population-Based Optimisation Algorithms

Ahmed Salah¹(✉), Emma Hart², and Kevin Sim²

¹ Faculty of Science, Mansoura University, Mansoura, Egypt
a.salah@mans.edu.eg

² School of Computing, Edinburgh Napier University, Edinburgh, UK
{e.hart,k.sim}@napier.ac.uk

Abstract. We describe a novel diversity method named Grid Diversity Operator (GDO) that can be incorporated into population-based optimization algorithms that support the use of *infusion* techniques to inject new material into a population. By replacing the random infusion mechanism used in many optimisation algorithms, the GDO guides the containing algorithm towards creating new individuals in sparsely visited areas of the search space. Experimental tests were performed on a set of 39 multimodal benchmark problems from the literature using GDO in conjunction with a popular immune-inspired algorithm (opt-ainet) and a sawtooth genetic algorithm. The results show that the GDO operator leads to better quality solutions in all of the benchmark problems as a result of maintaining higher diversity, and makes more efficient usage of the allowed number of objective function evaluations. Specifically, we show that the performance gain from using GDO increases as the dimensionality of the problem instances increases. An exploration of the parameter settings for the two main parameters of the new operator enabled the performance of the operator to be tuned empirically.

Keywords: Grid · Diversity · Optimization · Evolutionary algorithms · Artificial immune systems

1 Introduction

Managing the diversity of a population has been recognized as one of the most influential factors within an Evolutionary Algorithm (EA) right from their inception. From an exploration and exploitation perspective, an increase in diversity correlates with the exploration phase of an optimization algorithm whilst a decrease correlates with the exploitation phase. Maintaining a diverse population through the use of exploration operators is key to achieving a balance between the two phases [1].

The term *diversity* refers to differences among individuals which can be at either the genotype or phenotype level. Typically, diversity is classified into three categories: diversity maintenance, diversity control, and diversity learning. Diversity *maintenance* encourages exploring multiple promising pathways through the search space simultaneously in hopes of reaching higher fitness peaks. On the other hand, diversity *control* methods use population diversity, individual fitness, and/or fitness improvements as feedback to steer the evolutionary process towards exploration or exploitation. The main difference between diversity control and diversity learning methods is that in the former case, the short-term history (e.g., current population) is often used during diversity computation, whilst in the latter case, long-term history is used in combination with machine learning techniques to find (un)explored search areas. More recent research focuses on diversity *learning*, using cultural learning or self-organizing maps for example in order to discover promising locations within the search space.

In this work we extend research within the field of diversity *maintenance*. We describe a novel genotypic diversity learning method named Grid-Diversity-Operator (GDO) that makes use of the long-term history of all populations in order to suggest a biased distribution for new individuals. GDO is not specific to any particular algorithm, but can be used with any optimization algorithm that supports the use of *infusion* techniques — that is, insertion of new individuals after a certain number of generations or special initialization techniques. The new operator is tested within two population-based optimization algorithms that support infusion techniques, replacing the infusion step that causes random generation of new individuals. Experiments are conducted on a set of 13 benchmark multimodal optimization problems in different search space dimensions. We proposed an outline of this operator in a recent short paper [2]. Here we provide a full description of the operator and the motivation behind it. In addition, we undertake the first thorough evaluation of the operator using a well-known set of 39 multimodal benchmark problems, ranging in dimensionality from 2 to 30 (in contrast, the earlier work considered a small set of 2-dimensional functions only as proof-of-concept). Finally, we incorporate the GDO operator into two different optimisation algorithms to provide evidence that GDO is not tied to any particular algorithm, but can be used as a generic operator. A comprehensive statistical analysis is conducted to provide a more accurate explanation of GDO’s performance.

This paper is structured as follows. Section 2 will cover related work in the literature regarding diversity preserving techniques. The Grid Diversity Operator (GDO) will be introduced in detail in Sect. 3. Section 4 will briefly introduce the algorithms that will be used in the experimental analysis. The experimental protocol followed here with the benchmark problems adopted and the testing procedure is introduced in Sect. 5. The obtained results and discussion along with the statistical analysis will be presented in Sect. 6, while the final comments will be given in Sect. 7.

2 Related Work

It is widely accepted that maintaining high levels of diversity within a population greatly contributes to algorithm performance [3]. Recent and detailed survey of the area is given in by Črepinšek *et al.* [4] in which a taxonomy of approaches for managing exploration and exploitation within a search algorithm is defined. Three approaches are identified: diversity *maintenance*, diversity *learning* and diversity *control*.

Of most relevance to our work is the branch of this taxonomy that concerns methods for *diversity maintenance*, which the authors divide into two classes: non-niching and niching. Within the former class, four approaches are defined — population-based, selection-based, crossover/mutation-based and hybrid. Our interest lies within population-based methods, which are sub-classified into methods that vary population size, eliminate duplicates, infusion techniques and external archives. We provide a short overview of relevant work in the area of infusion-techniques, given that the GDO operator proposed in this paper is inspired by this — the reader is referred to the survey article for detailed examples of each of the other techniques.

Infusion techniques typically involve the insertion of new individuals into the population after a certain number of generations or managing initialisation of the population (e.g. [5]). Early approach to this were described by Grefenstette in [6] in which random immigrants are inserted into the population every generation, and Koza [7] whose ‘decimation’ algorithm replaced a random fraction of the population each generation. More recently, Koumoutsis and Katsaras [8] proposed a SawTooth GA that utilised a periodic re-initialization step at which a number of new, random individuals are added to the population. This algorithm introduced an operator that varied the population size over time — the population size is linearly decreased over a number of generations, then suddenly increased to its original value through the addition of random immigrants, hence the sawtooth name. This was shown to achieve both better population diversity and overall performance on a set of continuous optimization benchmarks compared to a standard GA.

In addition, our proposed approach has some similarities to previous diversity *learning* methods. This class of methods is less well-studied than the maintenance methods — the idea is that the long-term history of the population can be used in combination with a machine learning techniques to learn which areas of the space have not been explored. For example, Leung *et al.* [9] propose a history-driven evolutionary algorithm (HdEA), in which a binary space-partitioning tree (BSP) is used to record evaluated individuals and their associated fitness throughout the evolution process. A guided anisotropic search governs the search direction based on the history of the BSP tree which results in an individual being either exploitatively mutated towards the nearest historical optimum or randomly mutated if the individual is already a local optimum. Other learning methods [10] use self-organising maps (SOMs) to learn about unexplored areas of the space using information from the whole evolutionary history; this information is used to determine *novelty* and thus encourage exploration.

Another class of algorithms that also exploit variable population size with random immigration are the *aiNET* series of algorithms, first proposed in [11]. Steps intrinsic to the core algorithm maintain diversity by suppressing individuals that are genotypically similar (thus varying population size) while additionally adding a small number of new, randomly generated individuals to the population each iteration. Andrews and Timmis [12] further increased the diversity of aiNET with respect to function optimization through adding an immune-inspired mutation operator. De França *et al.* [13] evaluated the diversity mechanisms of opt-aiNET compared to fitness-sharing methods using a set of continuous optimization benchmarks, finding that the diversity mechanisms within opt-aiNET both obtained better solutions but also maintained more diverse solutions.

Most *infusion techniques* described in the literature rely on *random* initialisation methods; in contrast, we propose an operator that biases the generation of new individuals towards areas of the search space that are under-explored by the current population. The proposed GDO operator shares some similarities with previous learning approaches in that it exploits the history of the complete evolutionary process, however unlike [9] we focus in its use to guide exploration rather than exploitation. The operator can be used with any optimization algorithm that uses an infusion-technique. In this paper, we test its validity by replacing the random generation step of both Opt-aiNET and SawTooth algorithms which we will briefly introduce to give better understanding in Sects. 4.1 and 4.2.

3 Grid Diversity Operator (GDO)

The Grid Diversity Operator (GDO) is a novel approach for achieving exploration and exploitation balance through maintaining diversity. It can be defined as a hybrid, non-niching, population-based, genotype diversity maintaining and learning technique. Simply put, GDO is a special infusion technique for initializing new individuals that are inserted into a population after a certain number of generations. Instead of randomly initializing individuals over the whole domain, GDO tries to initialize them in unexplored locations. A memory archive is used to store information collected throughout the run regarding the distribution of the individuals, and is used to infer rarely visited locations.

The basic idea behind GDO is to split the feasible space (the domain) into smaller sub-spaces using the grid size parameter $G_{sz} \in \mathbb{R}^n$, which defines the number of intervals per dimension, where n is the number of dimensions for the problem. This process will form a 2D grid for 2-dimensional problem, 3D grid for a 3-dimension problem, and so on as demonstrated in Fig. 1. The GDO will then try to distribute new individuals into the grid slots that have received fewer visits over time, thus increasing the explorative power of the algorithm.

First, a memory archive is initialized as an empty dictionary that has n component *keys*, where each of them matches a single value. The *keys* refer to the indices of a slot within the grid, while the *value* represents the number of individuals that have previously been placed in this slot. The memory archive is designed in this manner for efficiency: as most of initialized memory will be

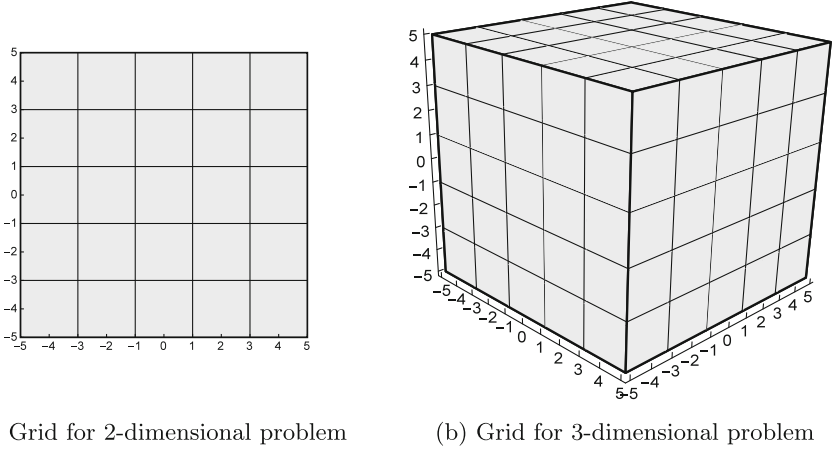


Fig. 1. Examples of grid for 2-dimensional and 3-dimensional problems

sparse, by using the dictionary only visited locations are stored, and therefore the use of memory is efficient regardless of problem dimensionality.

An example of the memory archive is shown in Table 1 that represent a sample of a memory archive with two entries for a 10-dimensional problem.

Table 1. Example of memory archive for 10-dimensional problem

Slot	Dictionary key (dimensions indexes)										N (value)
	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	d_{10}	
1	5	2	1	4	3	1	5	2	8	1	7
2	7	3	4	1	3	6	3	2	5	3	4

For the first entry, the slot located at key $< 5, 2, 1, 4, 3, 1, 5, 2, 8, 1 >$ has been assigned a value of 7, indicating that 7 solutions have appeared in this slot during the run so far. The slot located at key $< 7, 3, 4, 1, 3, 6, 3, 2, 5, 3 >$ has been assigned a value of 4 indicating 4 solutions visited this slot. For each iteration in the containing algorithm, every new individual is processed to identify its slot to update the memory archive. For each individual being processed, if there is an entry in the archive with a key that matches the identified slot, the value corresponding to this entry is increased by one. Otherwise, a new entry is added to the archive with a value of 1. The process of updating the memory archive is demonstrated in Listing 1.

After processing all individuals, the updated archive is used to initialize new individuals. For each new individual required, we pick a slot $S_{(d_1, d_2, \dots, d_n)}$ at random and calculate its *distribution probability* P according to Eq. 1.

$$P = e^{(-N)} \quad (1)$$

Listing 1. GDO Update Archive Method

Input: MemoryArchive, Individuals, Resolution
Output: MemoryArchive

```

for Individuali ∈ Individuals do
  Key ← FindSlot(Individuali)
  if KeyExist(MemoryArchive,Key) then
    IncreaseValue(MemoryArchive,Key,1)
  else
    AddKey(MemoryArchive,Key)
    SetValue(MemoryArchive,Key,1)
  end if
end for
Return (MemoryArchive)

```

where N is the value matching the slot key in the archive or zero if the slot does not yet belong to the archive. Finally, the calculated probability P is compared to the probability threshold parameter of the algorithm, P_{th} , and if $P > P_{th}$ then a new individual is initialized randomly in this specific slot. If not, another slot is picked at random, and the steps are repeated until the individual is initialized successfully. The Grid Diversity Operator, *GDO*, is described in Listing 2.

Listing 2. Grid Diversity Operator

Input: MemoryArchive, N_{new} , G_{sz} , P_{th}
Output: S_{new}

```

 $S_{new} \leftarrow \emptyset$ 
for  $i = 1$  to  $N_{new}$  do
  distributed ← FALSE
  while  $\neg$  distributed do
    Key ← PickSlotAtRandom( $G_{sz}$ )
    if KeyExist(MemoryArchive,Key) then
       $V \leftarrow \text{GetValueOfKey}(\text{MemoryArchive}, \text{Key})$ 
       $P \leftarrow e^{-V}$ 
    else
       $P \leftarrow 1$ 
    end if
    if  $P > P_{th}$  then
      Individual ← CreateNewIndividualInSlot(Key)
      InsertIndividual( $S_{new}$ , Individual)
      distributed ← TRUE
    end if
  end while
end for
Return ( $S_{new}$ )

```

The process of updating the memory archive and distributing new individuals continues until the algorithm terminates, at which point the final population is expected to be more diverse than simply using a random initialisation procedure.

4 Selected Infusion-Supported Algorithms for Experimentation

To assess the performance of the proposed Grid Diversity Operator (GDO), we incorporate it within two algorithms from the literature that feature a distinct

diversity maintenance mechanism. The first of the two is the artificial immune algorithm *opt-ainet* [11] and the other one is the *sawtooth* genetic algorithm [14].

4.1 Opt-aiNET Algorithm

Opt-aiNET, proposed in [11], is a well-known Artificial Immune System optimization algorithm. Opt-aiNET evolves a population that consists of a network of candidate solutions to the function being optimised known as antibodies. These undergo a process of evaluation against the objective function, clonal expansion, mutation, selection and interaction between themselves resulting in a population of dynamically changing size. Overtime, Opt-aiNET creates a memory set of antibodies that represent the best candidate solutions to the objective function. One of the main features of opt-ainet algorithm is that it has a defined, separated, diversity mechanism that injects a small number of new randomly created solutions into the population following each cycle of clonal expansion and mutation (the *AppendNewRandomCells()* shown in the bottom of Listing 3). This diversity-maintaining step makes it an ideal candidate for GDO injection.

Listing 3. Opt-aiNET Algorithm

Input: Initial_{size}, N_{clones}, Supp_{th}, Err_{th}, Mutation_{par}, Div_{ratio}
Output: Population
 Population $\leftarrow \emptyset$
 AppendNewRandomCells(Population, Initial_{size})
while \neg StopCondition() **do**
 EvaluateCells(Population)
 Clones \leftarrow CloneCells(Population, N_{clones}, Err_{th})
 MutatedClones \leftarrow HypermutateClones(Clones, Mutation_{par})
 EvaluateCells(MutatedClones)
 for Cell_i \in Population **do**
 BestClone \leftarrow GetBestClonePerCell(MutatedClones, i)
 if F(BestClone) < F(Population[i]) **then**
 Population[i] \leftarrow BestClone
 end if
 end for
 SupressLowAffinityCells(Population, Supp_{th})
 AppendNewRandomCells(Population, Div_{ratio})
end while
 Return (Population)

Most of the parameters of opt-ainet were set as suggested by the authors of the algorithm [11]. We set the number of clones per cell N_{clones} = 10, suppression threshold Supp_{th} = 0.2, error threshold Err_{th} = 0.001, mutation parameter μ = 100 and diversity ratio D_{rate} = 40 %. The only change was the value of initial population size which we set to 50 instead of the suggested value of 20. This change was important to help the algorithm with high dimensional problems and has been applied to all opt-ainet versions whether injected with GDO or not.

4.2 SawTooth Algorithm

The SawTooth Genetic Algorithm [14] is an algorithm that follows a sawtooth scheme defined by population size mean \bar{n} , an amplitude D and period of variation T in order to balance periods of exploration and exploitation. During every period, the population size decreases linearly. At the start of the next period, a population re-initialization occurs by appending randomly generated individuals to the population as shown in Listing 4. To achieve variable sized population, the population size is calculated every generation according to Eq. 2 below.

$$n(t) = \text{int} \left\{ \bar{n} + D - \frac{2D}{T-1} \left[t - T \cdot \text{int} \left(\frac{t-1}{T} \right) - 1 \right] \right\} \quad (2)$$

where $\text{int}(\cdot)$ is the floor function. The authors suggested a range for the optimum values of the T and D parameters of the sawtooth based on experimentation. The optimum normalized amplitude D/\bar{n} being from 0.9 to 0.96 and the optimum normalized period ranging from $T/\bar{n} = 0.5$ to 0.7. In this paper, we choose $\bar{n} = 80$, $T = 50$, $D = 75$ and these values complies with the ranges suggested by the algorithm authors. We also set the initial population size to 200, crossover rate to 0.7 and mutation rate to 0.95.

Listing 4. SawTooth Algorithm

Input: Population_{size}, CrossOver_{rate}, Mutation_{rate}, \bar{n} , D , T
Output: Population

```

t ← 0
Population ← InitializePopulation(Populationsize)
EvaluatePopulation(Population)
while ¬ StopCondition() do
  t ← t + 1
  n(t) ← CalculateNextPopulationSize( $\bar{n}$ ,  $D$ ,  $T$ , t)
  if mod(t,  $T$ ) = 0 then
    NewIndividuals ← InitializePopulation(2 *  $D$ )
    InsertIndividuals(Population, NewIndividuals)
  end if
  NextPopulation ← ∅
  for i = 1 to n(t) by 2 do
    [P1, P2] ← SelectParents()
    [C1, C2] ← Crossover(P1, P2, CrossOverrate)
    Mutate(C1, Mutationrate), Evaluate(C1)
    Mutate(C2, Mutationrate), Evaluate(C2)
    [B1, B2] ← SelectBestTwoIndividuals([P1, P2, C1, C2])
    InsertIndividuals(NextPopulation, [B1, B2])
  end for
  Population ← NextPopulation
end while
Return (Population)

```

5 Experimental Protocol

The aim of the experiments is to assess if the performance of both algorithms introduced above improves when injected with GDO in order to determine whether GDO can help in achieving better quality solutions.

In addition, since GDO introduces two parameters – grid size G_{sz} and probability threshold P_{th} , we investigate settings for each parameter in a brief empirical investigation. For the grid size parameter, G_{sz} , the chosen values were (100, 500, 1000) while for the probability threshold P_{th} the values were (0.01, 0.10, 0.20). Thus, nine configurations of GDO are tested in an attempt to find the most successful configuration as shown in Table 2.

Table 2. Configuration code names and parameter values

	GDO-1	GDO-2	GDO-3	GDO-4	GDO-5	GDO-6	GDO-7	GDO-8	GDO-9	GDO-free
G_{sz}	100	100	100	500	500	500	1000	1000	1000	N/A
P_{th}	0.01	0.10	0.20	0.01	0.10	0.20	0.01	0.10	0.20	N/A

The experimental tests are performed using opt-ainet and sawtooth algorithms on a set of benchmark problems from the *CEC 2014* benchmark suite [15]. The selected problems (F_4 through F_{16}) are multimodal problems where many of which have huge numbers of local optima that make the assessment process challenging. The rest of the problems from the suite (three unimodal functions F_1 – F_3 and 14 hybrid/composite functions F_{17} – F_{30}) are interesting as well but have different features and therefore were not selected.

Following the evaluation criteria defined in *CEC 2014* benchmark suite [15], both algorithms with and without GDO (including the nine GDO configurations) will be tested against the 13 problems in 2, 10 and 30 dimensional space. For every function, each algorithm/configuration is run 25 times with a maximum number of function evaluations equal to (Max_{FES}) of $10,000 * D$ where D is the number of dimensions. Therefore, Max_{FES} is 20,000 for two dimensions, 100,000 for 10 dimensions and 300,000 for 30 dimensions. The best quality solution is noted for each run, along with the number of function evaluations at which the best solution is found: if this is less than Max_{FES} this indicates stagnation of the algorithm. A detailed statistical analysis is conducted to assess the results.

6 Results and Discussion

Due to space limitations, it is not possible to provide detailed tables for the results of all experiments with 39 test cases (13 problems over three different dimensions) for both algorithms with 10 configurations each — a total of 780 results. In an attempt to summarize the data, we count the number of problem instances in which a GDO configuration outperforms the corresponding GDO-free algorithm. Table 3 displays this information for the nine GDO configurations used within the two algorithms over the three different dimensions for the benchmark problems. For instance, the first entry denotes that opt-ainet injected with GDO-1 outperformed GDO-free opt-ainet in 5 problems out of 13 instances with respect to solution quality (while in the remaining 8 problems either GDO-free was better or there were no significant difference). Two observations are clear

from Table 3. Firstly, the number of instances in which GDO outperforms GDO-free increases as the dimensionality of the instances increases, i.e., its benefit increases with dimensionality. Secondly, the GDO-9 configuration performs best out of the 9 different parameterisations, when considered across all problem dimensions.

Table 3. Number of functions (out of 13) where a GDO configuration achieves better quality solutions than GDO-free

Config.	Dimension [Opt-aiNET]			Dimension [SawTooth]		
	2	10	30	2	10	30
GDO-1	5	8	12	7	7	11
GDO-2	8	8	11	9	8	12
GDO-3	6	8	11	9	9	8
GDO-4	5	6	10	6	10	11
GDO-5	8	7	10	5	10	9
GDO-6	9	8	10	8	9	9
GDO-7	8	5	12	7	9	10
GDO-8	9	6	11	7	5	10
GDO-9	10	9	12	9	9	10

Since the algorithm/configuration that optimises both objectives (minimises convergence speed and minimises summed fitness) is always considered more successful, additionally we provide graphs that summarise the results concerning both solution quality and convergence speed.

Figures (2-7) summarise the results where every graph contains the result of 9 GDO configurations and one without GDO injection (GDO-free) for a specific algorithm and problem dimensionality. The different configurations are distributed on the graph’s horizontal axes. The bars on top denote the sum of the normalized fitness for the configuration over the 13 problem instances; similarly, the bars on the bottom refer to the sum of the normalized of number of evaluations used before algorithm termination. Error bars are provided as an indication of difference in means between configurations for both quality (top) and efficiency (bottom).

Beginning with the optainet algorithm results, we see in Fig. 2 that all GDO-injected configurations of opt-ainet for two-dimensional problems outperform the GDO-free version with respect to solution quality. However, all the GDO configurations use more evaluation cycles than the GDO-free version. For the ten-dimensional tests, Fig. 3 shows that 7 out of 9 of the GDO configurations of opt-ainet — all but GDO-7 and GDO-8 — surpasses the GDO-free version in terms of solution quality. In this case, GDO-4, GDO-5, and GDO-8 configurations used less number of function evaluations than the GDO-free version while GDO-1, GDO-3, GDO-6 were similar to GDO-free and the rest were the worst. Figure 4

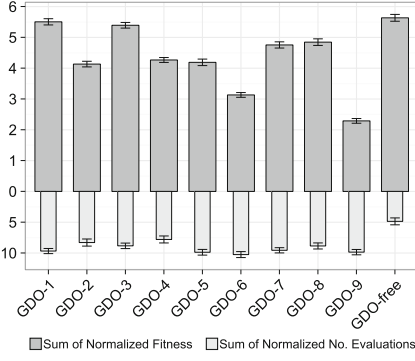


Fig. 2. Opt-aiNET (2 Dimensions)

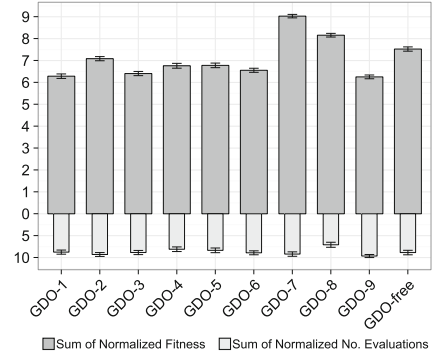


Fig. 3. Opt-aiNET (10 Dimensions)

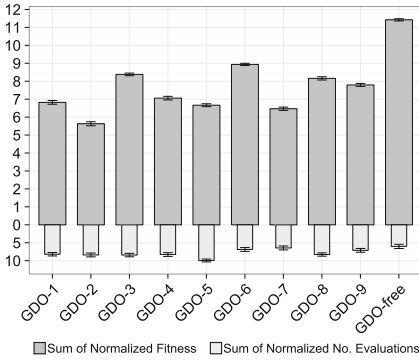


Fig. 4. Opt-aiNET (30 Dimensions)

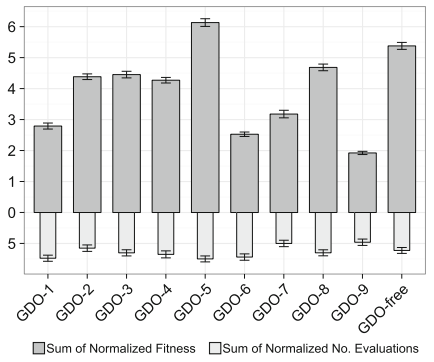


Fig. 5. Sawtooth (2 Dimensions)

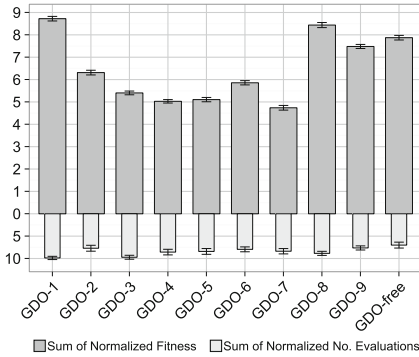


Fig. 6. Sawtooth (10 Dimensions)

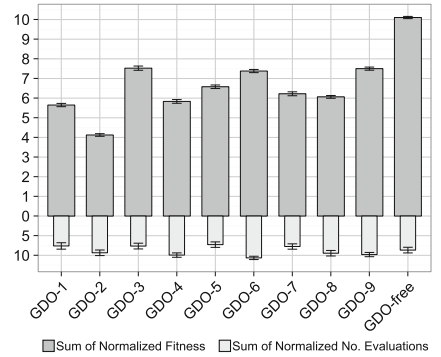


Fig. 7. Sawtooth (30 Dimensions)

shows the results for the 30-dimensional tests where we see that all instances of GDO configured opt-ainet algorithm were able to achieve better results than GDO-free opt-ainet, while all GDO configurations have utilised more function evaluations than GDO-free.

The sawtooth algorithm for two-dimensional problems results (Fig. 5) shows that only the GDO-5 configuration failed to achieve better quality solutions than GDO-free sawtooth. In this case, GDO-2, GDO-7 and GDO-9 configurations shown faster convergence while the rest were the worst with respect to number of evaluations used. The results of the ten-dimensional sawtooth tests are shown in Fig. 6 were seven GDO-injected sawtooth algorithm instances outperform the GDO-free version for quality. However, all GDO versions utilise more evaluation cycles in this case. Finally, for the 30-dimensional tests, the results in Fig. 7 shows that all GDO instances outperformed the GDO-free version with respect to solution quality while GDO configurations (GDO-1, GDO-3, GDO-5 and GDO-7) shown better utilisation to their allowance of function evaluations than GDO-free version of sawtooth.

It is interesting to note that some GDO configurations are able to achieve better results with fewer evaluations than GDO-free, e.g. in Figs 3, 5 and 7. However, these are exceptions: in general, the GDO configurations utilise more function evaluations than the GDO-free configurations for both opt-ainet and sawtooth algorithms due to the additional exploration ability facilitated by the GDO operator.

In summary, in terms of solution quality, the majority of GDO configurations provide an improvement in all the six experiments defined by (*algorithm/dimensionality*). Out of them we note GDO-9 which performed better than the rest of the configurations in three instances as shown in Figs 2, 3 and 5 followed by GDO-2 as a second best beating all configurations in two other instances as shown in Figs 4 and 7. The benefit increases with dimensionality, with GDO-9 winning 12/13 instances when used with opt-ainet, and 10/13 cases when used with the saw-tooth algorithm for 30 dimensions problems according to Table 3. To further demonstrate the performance of GDO-9 configuration, Tables 4 and 5 shows the median for all the runs over the 13 functions using the algorithms opt-ainet and sawtooth respectively. Comparing the results of GDO-free in both tables we can see that sawtooth capable of achieving better results than opt-ainet for all problems especially in higher dimensions where opt-ainet tends to diverge notably with functions 4 and 15. For function 15, and while GDO-9 helped sawtooth to make significant improvements for the same function for all dimensions, GDO-9 help for opt-ainet was much more important and the performance drastically improved over the poor results of GDO-free opt-ainet especially in 10 and 30 dimensions.

With respect to number of evaluations, it is clear that most GDO-injected algorithms require more function evaluations to converge than the GDO-free version of the algorithm. Recall that each algorithm is given a fixed number of function evaluations (Max_{FES}), but terminates before this in case of no improvement which in many cases reflect being stuck in a local optima. This is clearly

Table 4. Median of Opt-aiNET algorithm results: GDO-free against GDO-9 over the 3 dimensions for the 13 functions

Fn	Dimension [GDO-free]			Dimension [GDO-9]		
	2	10	30	2	10	30
4	400.4180170	15076.92619	41522.53368	400.0067050	1382.623949	25880.42417
5	510.1203580	520.2005640	520.6004260	504.8840650	520.0004160	520.2967840
6	601.0638150	617.8945810	650.5898950	600.3518890	611.1999250	643.6837470
7	713.9159790	1404.468688	2867.517032	700.1278140	806.8315380	1676.329282
8	802.0770040	1012.395469	1285.506462	800.3384240	894.4034050	1244.529391
9	901.2102780	1154.560341	1436.587623	901.8637690	1000.095998	1437.381565
10	1092.261931	4307.027543	10982.91621	1014.131418	2713.651994	9295.601940
11	1222.144605	4525.313885	13180.02786	1114.076847	3326.207438	9469.204918
12	1200.005679	1202.956275	1206.578518	1200.004737	1201.733634	1203.362254
13	1301.264216	1307.748503	1309.748985	1300.567841	1303.354419	1308.982133
14	1400.341883	1612.230827	1783.065834	1400.100778	1435.029170	1728.472052
15	1500.167713	3556487.517	98467228.88	1500.048914	4426.678862	7485.869758
16	1600.156432	1604.219537	1614.369001	1600.156378	1603.941531	1613.563775

Table 5. Median of SawTooth algorithm results: GDO-free against GDO-9 over the 3 dimensions for the 13 functions

Fn	Dimension [GDO-free]			Dimension [GDO-9]		
	2	10	30	2	10	30
4	400.0142740	727.9746490	9170.752282	400.0065010	651.3681520	8160.968453
5	507.9617110	520.0003550	520.0021610	514.0684370	520.0002660	520.0022920
6	600.1977440	611.0157390	641.2174190	600.1320940	610.4406700	639.3822960
7	700.9761200	759.2174120	1189.677979	700.2440890	769.6810190	1157.155085
8	800.0000000	838.8033260	944.6040920	800.9949590	827.8587750	946.8881300
9	901.9899180	941.7881410	1094.533675	900.9949590	950.7426880	1093.022542
10	1000.312173	1140.325716	4056.051603	1000.312173	1269.169192	3983.343406
11	1218.438335	2277.064899	5145.395051	1218.438335	2454.652439	5542.767461
12	1200.000579	1200.698797	1201.134650	1200.000604	1200.664882	1201.048815
13	1300.062153	1303.208397	1305.765346	1300.018911	1303.365590	1306.059232
14	1400.004992	1404.043086	1570.808730	1400.125393	1405.069313	1567.143899
15	1500.090571	1952.314124	2576.155155	1500.019729	1948.094013	2391.734321
16	1600.356445	1603.382337	1612.343996	1600.074448	1603.331906	1612.063572

observed in Figs 2 and 4 that show that the GDO-free algorithms stagnates early, therefore recording a low number of function evaluations, but has the worst solutions from the quality perspective. The larger number of evaluations used by GDO in the majority of experiments reflects the extra effort used by the operator to continue searching in novel parts of the solution space, therefore using the allowed budget of evaluations to avoid convergence to local optima. Thus in general GDO facilitates longer running times, leading to improved solution quality, though clearly there is computational cost to this in terms of utilised CPU time.

6.1 Statistical Analysis

Although the graphical summaries given above provide some insight and intuition regarding GDO performance, a proper statistical analysis is required to justify any claims. The statistical analysis in this section addresses the following questions:

- Does GDO injection helps improve solution quality when compared to the equivalent GDO-free algorithm?
- Which GDO configuration (from the proposed nine configurations) shows the best performance in terms of solution quality?

Since we have two algorithms, each with 10 possible configurations (9 GDO, 1 GDO-free) and each is tested against 13 problems in each of three different dimensions, there are 780 datasets. Each dataset contains the results of running a specific algorithm/configuration on a specific problem and specific dimension 25 times.

Before conducting any statistical analysis of the results, we utilise a Shapiro-Wilk normality test (with a level of significance of $\alpha = 0.05$) to determine with the datasets follow a normal distribution in order to identify appropriate statistical tests. The samples in 194 groups out of 780 were found to appear normally distributed while 586 groups were not and therefore non-parametric tests are used.

We used the non-parametric MannWhitney U test (with a level of significance of $\alpha = 0.05$) to test for difference in means and Table 6 shows the number of cases (out of 78) where a GDO configuration outperformed GDO-free and those where GDO-free dominated. The test shows that if there is a statistically difference in solution mean between GDO-injected and GDO-free algorithm, then the probability that the GDO-injected algorithm will lead to better quality solutions is twice that of the GDO-free one: the final row of Table 6 shows that GDO configurations are better in 158 cases, as opposed to 75 for the GDO-free. In addition, the results for the GDO-9 configuration across all experiments appear to be statistically better with respect to solution quality than any other GDO configuration when compared to GDO-free algorithms results. As shown in Table 6, the probability that GDO-9 will likely have better quality solutions is three times that of the GDO-free algorithm.

Table 6. MannWhitney U test results (aggregated)

Config	GDO-1	GDO-2	GDO-3	GDO-4	GDO-5	GDO-6	GDO-7	GDO-8	GDO-9	Total
Better	14	18	15	19	17	22	16	19	18	158
Worse	9	9	9	9	9	9	8	7	6	75

7 Conclusion

Diversity control methods for search algorithms fall into three classes: diversity maintenance, diversity learning and diversity control. We have described a new operator, GDO, that directs exploration into previously unvisited areas of the search space. While mainly falling into the class of diversity maintenance algorithms, it also draws inspiration from diversity learning methods in making use of historical information from the evolution process.

The GDO operator was shown to achieve effective exploration through testing on the benchmark problems when incorporated within opt-ainet and sawtooth algorithms. When compared to the GDO-free versions of the algorithms, it was shown that GDO can significantly help a supported algorithm to achieve better quality solutions in most cases. Importantly, Table 3 showed that the benefit of the GDO operator increases as the dimensionality of the problems increases. It was noticeable however that GDO-injected algorithms tend to use more function evaluation cycles than the GDO-free versions, i.e. does not stagnate prematurely. GDO forces the algorithm to explore more of the space thus using more evaluations while exploring but without preventing the algorithm's internal exploitation method from finding improved solutions. This added exploration ability helps the injected algorithm to both escape local optima and to achieve better solutions.

Within the current implementation, the grid size is fixed for all dimensions such that setting the grid size parameter $G_{sz} = 100$ in three-dimensional problem will correspond to a grid resolution of $100 \times 100 \times 100$. There is no obligation to set the grid resolution equally for all dimensions and in fact, it may be better to set different values depending on the dimensionality of the problems. In this paper, we empirically studied the values of grid size and probability threshold parameters of GDO by selecting three appropriate values for each. Although the nine proposed configurations of GDO (each with different parameters values) behaved differently, certain configurations were found able to offer the best possible performance across the tests, in particular the GDO-9 configuration where $G_{sz} = 1000$ and $P_{th} = 0.20$. The GDO-9 configuration was able to outperform GDO-free in all cases as shown in Figs 2–7 and all other GDO configurations as well in three cases as shown in Figs 2, 3 and 5. In addition, the statistical analysis conducted in this paper confirmed GDO-9's superiority. However, by incorporating feedback from search process it seems clear that this could be achieved autonomously, so that both parameters in fact could adapt over a single run to achieve better performance. We aim to address this issue in future work.

References

1. Soza, C., Becerra, R.L., Riff, M.C., Coello Coello, C.A.: Solving timetabling problems using a cultural algorithm. *Appl. Soft Comput.* **11**(1), 337–344 (2011)
2. Salah, A., Hart, E.: Grid diversity operator for some population-based optimization algorithms. In: *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference*, pp. 1475–1476. ACM (2015)

3. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*. Springer, New York (1996)
4. Črepinšek, M., Liu, S., Mernik, M.: Exploration and exploitation in evolutionary algorithms. *ACM Comput. Surv.* **45**(3), 1–33 (2013)
5. Li, Z., Wang, X.: Chaotic differential evolution algorithm for solving constrained optimization problems. *Inf. Technol. J.* **10**, 2378–2384 (2011)
6. Grefenstette, J.J.: *Genetic Algorithms for Changing Environments*. Elsevier, Amsterdam (1992)
7. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge (1992)
8. Koumoussis, V.K., Katsaras, C.P.: A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Trans. Evol. Comput.* **10**(1), 19–28 (2006)
9. Leung, S.W., Yuen, S.Y., Chow, C.K.: Parameter control by the entire search history: case study of history-driven evolutionary algorithm. In: *2010 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, July 2010
10. Amor, H.B., Rettinger, A.: Intelligent exploration for genetic algorithms: using self-organizing maps in evolutionary computation. In: *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, GECCO 2005*, pp. 1531–1538. ACM, New York (2005)
11. de Castro, L.N., Timmis, J.: An artificial immune network for multimodal function optimization. In: *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC 2002*, vol. 1, pp. 699–704 (2002)
12. Andrews, P.S., Timmis, J.: On diversity and artificial immune systems: incorporating a diversity operator into aiNet. In: Apolloni, B., Marinaro, M., Nicosia, G., Tagliaferri, R. (eds.) *WIRN 2005 and NAIS 2005*. LNCS, vol. 3931, pp. 293–306. Springer, Heidelberg (2006)
13. De França, F.O., Coelho, G.P., Zuben, V., F.J.: On the diversity mechanisms of opt-aiNet: a comparative study with fitness sharing. In: *IEEE World Congress on Computational Intelligence, WCCI 2010–2010 IEEE Congress on Evolutionary Computation, CEC 2010* (2010)
14. Koumoussis, V., Katsaras, C.: A Saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Trans. Evol. Comput.* **10**(1), 19–28 (2006)
15. Liang, J.J., Qu, B.Y., Suganthan, P.N.: Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. Technical report, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China, December 2013

Applications of Evolutionary Computation
19th European Conference, EvoApplications 2016,
Porto, Portugal, March 30 -- April 1, 2016, Proceedings,
Part II
Squillero, G.; Burelli, P. (Eds.)
2016, XXVI, 329 p. 94 illus., Softcover
ISBN: 978-3-319-31152-4