An Evolutionary Algorithm with Species-specific Explosion for Multimodal Optimization

Ka-Chun Wong, Kwong-Sak Leung, Man-Hon Wong
Department of Computer Science & Engineering
The Chinese University of Hong Kong, Hong Kong
{kcwong, ksleung, mhwong}@cse.cuhk.edu.hk

ABSTRACT

This paper presents an evolutionary algorithm, which we call Evolutionary Algorithm with Species-specific Explosion (EASE), for multimodal optimization. EASE is built on the Species Conserving Genetic Algorithm (SCGA), and the design is improved in several ways. In particular, it not only identifies species seeds, but also exploits the species seeds to create multiple mutated copies in order to further converge to the respective optimum for each species. Experiments were conducted to compare EASE and SCGA on four benchmark functions. Cross-comparison with recent rival techniques on another five benchmark functions was also reported. The results reveal that EASE has a competitive edge over the other algorithms tested.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*Unconstrained optimization*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms

Keywords

Evolutionary Algorithm, Genetic Algorithm, Multimodal Optimization, Function Optimization, Species-specific Explosion, Species Conserving Genetic Algorithm

1. INTRODUCTION

Given an optimization problem, traditional optimization algorithms can be applied to obtain the global optimum. However, in the real world, we are often interested in not only a single global optimum, but also other possible global and local optima. Such a requirement imposes a great challenge for researchers to apply traditional optimization algorithms to solve the problem. Nevertheless, evolutionary algorithms [6] are particularly effective in solving it [4].

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

To trace back the history, the work of Kenneth De Jong [7] is the first known attempt to solve multimodal optimization problems using evolutionary algorithms. He introduced the crowding technique to increase the chance for locating multiple optima. In the crowding technique, an offspring replaces the parent which is most similar to the offspring itself. Such a strategy can preserve the diversity and maintain different types of individuals in a run. Twelve years later, Goldberg and Richardson [5] proposed a fitness-sharing niching technique as a diversity preserving strategy to solve multimodal optimization problems. He proposed to use a shared fitness function, instead of an absolute fitness function, to evaluate the fitness of an individual in order to favor the growth of the individuals which are distinct to others. With this technique, a population can be prevented from the domination of a particular type of individuals. Since then, many researchers started to explore different ways to deal with the problems. These methods include: species conserving [10], crowding [7, 13], elitism [9], differential evolution [18], clearing [15], repeated iterations [1] and island model [2]. Though different methods were proposed in the past, they were all based on the same fundamental idea. It is to strike an optimal balance between convergence and diversity of evolutionary algorithm in order to locate all optima (global and local)

The species conserving technique for multimodal optimization was proposed by Li et al. [10]. It was claimed that the technique was considered as an effective and efficient method for inducing niching behavior into GAs. However, in our experiments, we find that the performance of the technique still has space for improvement. It always suffers from genetic drifts though each species is conserved with one individual. The result of the comparison test conducted by Singh et al. [16] also reveals that the species conserving technique performs the worst among the algorithms tested. As a result, we propose a novel algorithm to remedy the species conserving technique in this paper.

2. BACKGROUND

2.1 Species Conserving Genetic Algorithm

Species conserving genetic algorithm(SCGA) [10] is a technique for evolving parallel subpopulations for multimodal optimization. Before each generation starts to crossover, the algorithm selects a set of species seeds which can bypass the subsequent procedures and be saved into the next generation. The algorithm first divides a population into several species based on a dissimilarity measure. The fittest indi-

vidual is selected as the species seed for each species. After the identification of species seeds, the population undergoes the usual genetic algorithm operations: selection, crossover and mutation. As the operations may remove the survival of less fit species, the saved species seeds are copied back to the population at the end of each generation. The whole structure of SCGA is shown in Algorithm 1.

To determine the species seeds in a population, the algorithm first sorts the population in a decreasing fitness order. Once sorted, it picks up the fittest individual as the first species seed and forms a species region around it. The next fittest individual is tested on whether it is located in a species region. If not, it is selected as a species seed and another species region is created around it. Otherwise, it is not selected. Similar operations are applied on the remaining individuals, which are subsequently checked against all existing species seeds.

To copy the species seeds back to the population after the genetic operations have been executed, the algorithms need to scan all the individuals in the current population and identify to which species they belong. Once it is identified, the algorithm replaces the worst individual (lowest fitness) with the species seed in a species. If no individuals can be found in a species for replacement, the algorithm replaces the worst and un-replaced individual in the whole population. In short, the main idea is to preserve the population diversity by preserving the fittest individual for each species.

Algorithm 1 Species Conserving Genetic Algorithm

$G(t)$: Generation at time t
X_s: A set storing species seeds

$t \leftarrow 0$;
Initialize $G(t)$;
Evaluate $G(t)$;
while not termination condition **do**
  Identify Species Seeds X_s;
  Select $G(t+1)$;
  Crossover $G(t+1)$;
  Mutate $G(t+1)$;
  Evaluate $G(t+1)^{\ddagger}$;
  Conserve species from X_s in $G(t+1)$;
  $t \leftarrow t+1$;
end while
Identify species seeds X_s;
Identify global optima;

3. EVOLUTIONARY ALGORITHM WITH SPECIES-SPECIFIC EXPLOSION

Evolutionary Algorithm with Species-specific Explosion (EASE) is an evolutionary algorithm which identifies and exploits species seeds to locate global and local optima. There are two stages in the algorithm: **Exploration Stage** and **Species-specific Stage**. The exploration stage targets for roughly locating all global and local optima. It not only undergoes normal genetic operations: selection and crossover, but also involves the addition of randomly generated individuals for preserving the diversity. On the other hand, the species-specific stage targets for gently locating the optimum for each species. Species-specific genetic operations are applied. Only the individuals within the same species are allowed to perform selection and crossover to each other. No inter-species selection and crossover are allowed. Such a strategy is to provide more chances for each species to converge to its respective optimum, with the trade-off that diversity is no longer preserved. To have a better global picture for locating optima, EASE starts with the exploration stage. It will switch to the species-specific stage only after the stage switching condition is satisfied. No matter in which stage, a local operation called **Species-specific Explosion** is always executed so as to help species to climb and converge to its corresponding optimum. The whole structure of EASE is shown in Algorithm 2.

Algorithm 2 Evolutionary Algorithm with Species-specific Explosion

$G(t)$: Generation at time t
X_s: A set storing species seeds
E_s: A set storing species-specific exploded individuals
pop_size: Initial population size
K: A real number over the interval $[0, 1]$
SS: Species Specific stage switching parameter
$EMSS$: Expected Mutation Step Size

$t \leftarrow 0$;
$SS \leftarrow false$;
$EMSS \leftarrow$ mutation probability $\times$ mutation step size;
$pastIndividuals \leftarrow \emptyset$;
Initialize $G(t)$;
Evaluate $G(t)$;
while not termination condition **do**
  if $SS = false$ **then**
    Select $G(t+1)$;
    Crossover $G(t+1)$;
  else
    Species-specific Select $G(t+1)$;
    Species-specific Crossover $G(t+1)$;
  end if
  Mutate $G(t+1)$;
  Evaluate $G(t+1)^{\ddagger}$;
  $X_s \leftarrow$ IdentifySpeciesSeeds$(G(t+1))$;
  DeltaEval$(X_s, pastIndividuals)$;
  if $SS = false$ **then**
    $SS \leftarrow$ isSpeciesSpecific$(X_s, EMSS)$;
  end if
  $E_s \leftarrow$ SpeciesSpecificExplosion$(G(t+1), X_s, K, SS)$;
  $pastIndividuals \leftarrow X_s \cup E_s$;
  $G(t+1) \leftarrow X_s \cup E_s$;
  if $SS = false$ **then**
    Fills $G(t+1)$ with randomly generated individuals to reach pop_size;
  end if
  $t \leftarrow t+1$;
end while
Identify species seeds X_s;
Identify global optima;

3.1 Species Identification

To determine species in a population, we adopt the dissimilarity measurement proposed in Goldberg and Richardson

‡It involves Survival Selection if the generation is overlapping.

[5] and Li et al. [10]. The dissimilarity between two individuals are based on their Euclidean distance. The smaller the distance, the more similar they are:

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n}(x_{ik} - x_{jk})^2}$$

where x_i and x_j are two individuals, which are n-dimensional vectors $[x_{i0}, x_{i1}, ..., x_{in}]$ and $[x_{j0}, x_{j1}, ..., x_{jn}]$ respectively.

Each species is a subset of population. The fittest individual within a species is chosen as the species seed. The region around a species seed forms its corresponding species region. All individuals are classified as the same species if it is within the species distance (r_s) from the species seed. Petrowski [15] and Li et al. [10] proposed an algorithm to identify species seeds. The algorithm first sorts the population in a decreasing fitness order. Then it picks up the individual with the highest fitness as the first species seed and forms a species region around it. All individuals within r_s distance from the species seed are classified as the same species as that of the seed. For the next individual, it is checked whether it is within r_s distance from the species seed. If not, it is selected as another species seed. Similar operations are applied on the remaining individuals. Each individual is tested on whether it lies in others' species regions. If not, it is selected as a species seed. Otherwise, it is not selected. The main idea is to pick up the fittest individuals as the species seed for each species. The algorithm is shown in Algorithm 3.

Algorithm 3 Identify Species Seeds

```
procedure IdentifySpeciesSeeds(G)
    Sort G in decreasing fitness values;
    X_s ← ∅;
    while not reaching the end of G do
        Get best un-scanned individual i_s from G;
        found ← false;
        for ∀x ∈ X_s do
            if d(x, i_s) ≤ r_s then
                found ← true;
                break;
            end if
        end for
        if notfound then
            X_s ← X_s ∪ i_s;
        end if
    end while
    return X_s;
end procedure
```

3.2 Species Seed Delta Evaluation

After we have identified all species seeds in the population, we perform delta evaluation to record the recent step changes that can increase fitness for each species seed. For each species seed, we pick up the fittest individual of the same species in the previous generation, under the constraint that its fitness is lower than that of the species seed itself. By doing so, we can select the individual which is most likely the ancestor of a species seed in the previous generation. We call this individual as the Likelihood Ancestor(LA). If we can pick up the corresponding LA for a species seed, we store the value difference between the genome of LA and the genome of the species seed into the array *delta* of the species seed. Thus the array *delta* of a species seed serves as a memory recording the last known step sizes, which improved the species seed itself. The algorithm is shown in Algorithm 4. (All elements of *delta* are initialized to the mutation step size at the beginning)

Algorithm 4 Species Seed Delta Evaluation

```
dim: The maximum dimension
x.value[i]: The genome value of x at dimension i
LA.value[i]: The genome value of LA at dimension i

procedure DeltaEval(X_s, pastIndividuals)
    for ∀x ∈ X_s do
        LA ← the individual ∈ pastIndividuals
        with the highest fitness in the same species
        where its fitness is lower than that of x and x ≠ LA;
        if LA ≠ null then
            for i from 1 to dim do
                x.delta[i] ← x.value[i] − LA.value[i];
            end for
        end if
    end for
end procedure
```

3.3 Stage Switching Condition

To ensure a proper condition for switching from the exploration stage to the species-specific stage, we propose using the expected mutation step size ($EMSS$) as a measure for controlling the switching:

$$EMSS = p_m \times r_m$$

where p_m is the mutation probability and r_m is the mutation step size.

For each species seed, we scan its array *delta* to check whether its element exceeds $EMSS$. If there does not exist an element which exceeds $EMSS$, the switching condition is satisfied. The algorithm will switch to the species-specific stage. Otherwise, the algorithm will remain in the exploration stage. The rationale behind the checking condition is that $EMSS$ can give us an expected value for measuring the mutation ability of the algorithm. It can serve as a measurement to assess the ability of the algorithm to jump from one region to another region by just using mutation. Thus if all the elements of the arrays *delta* of all species seeds do not exceed the $EMSS$, it is reasonable to deduce that the fitness improvement steps for all species in the subsequent generations are mostly upper-bounded by the $EMSS$. All fitness improvement steps can be completed by merely using mutations, but not inter-species crossovers. Hence inter-species crossovers are no longer needed. Species-specific stage should be launched. The algorithm is shown in Algorithm 5.

3.4 Species-specific Explosion

In SCGA, Li et al. [10] proposed conserving one individual for each species. However, just one individual for each species is not enough for the algorithm to well-conserve and nurture the species. In a run of SCGA, it is often the case that the algorithm does conserve species with low fitness values, but they are present in a small proportion. It is because

Algorithm 5 Stage Switching Condition

dim: The maximum dimension

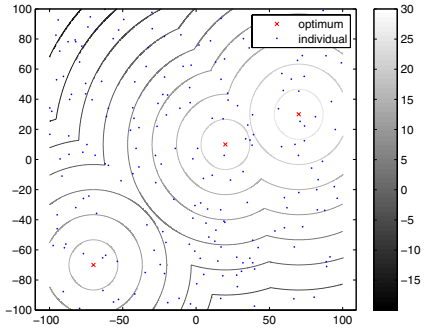
procedure isSpeciesSpecific($X_s, EMSS$)
  $SS \leftarrow true$;
  for $\forall x \in X_s$ **do**
    for i from 1 to dim **do**
      if $x.delta[i] \geq EMSS$ **then**
        $SS \leftarrow false$;
      end if
    end for
  end for
  return SS;
end procedure

once they form new offspring, their offspring are often removed quickly in subsequent generations due to their low fitness values. Thus most individuals are always of the species with high fitness values. An example is depicted in Figure 1. In the example, we can observe that the individuals gradually converge to the three optima fitness-proportionally. Though different species are preserved with an individual as the species seed, it cannot converge to the local optimum located at the left-bottom corner due to the relatively low fitness values there. Merely SCGA itself actually cannot provide enough indiscriminate condition for species to evolve and converge to its respective optimum in each run. Hence we propose a local operation called **Species-specific Explosion** to remedy the convergences in this paper.

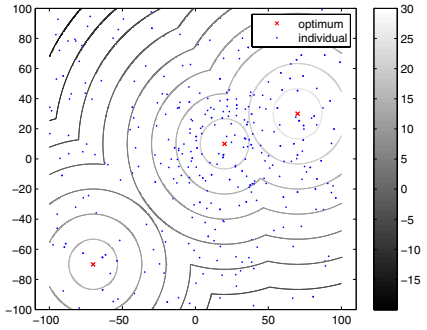
Species-specific explosion is the local operation in which we create multiple copies for each species seed and mutate them. To start this local operation, the algorithm needs to determine two parameters:

1. How many copies should be created for each species seed?
2. What is the mutation step size for each species seed?

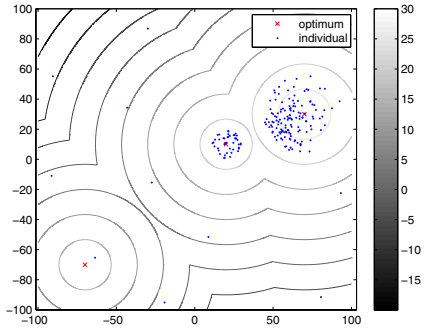
For the first question, we propose using the ratio of individuals in the same species to the current population to determine the number of copies to be created. The details are given in the section 3.5. For the second question, we propose using the array *delta* of species seed as the corresponding mutation step size. Recall that the array *delta* of species seed saves the step size values which were known to improve the species seed itself in the previous generation, it can be used for approximating how far the species seed should mutate to have a better fitness in the current generation. Hence we choose to use the array *delta* as the mutation step size of the species-specific explosion operation. Once the two parameters are calculated, the algorithm starts to check whether the species seed is present in the previous generation. If it is present, the algorithm will "explode" it, which means creating multiple copies and mutating them. Otherwise, no actions will be executed. The rationale behind the checking is to ensure that the species seed to be exploded is a stable species seed. Hence we require the species seed at least survive through one generation to be eligible for the explosion. Alternatively, if the current stage is species-specific stage, the above checking is overridden. All species seeds are eligible for the explosions, in order to provide all species an indiscriminate condition to evolve in this stage. The algorithm is shown in Algorithm 6.



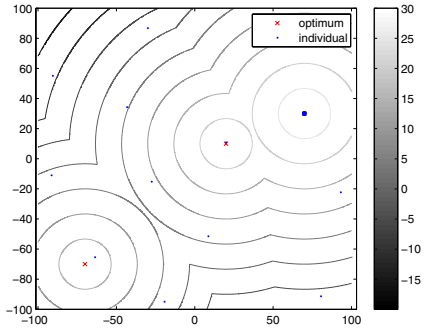
(a) Generation 1



(b) Generation 2



(c) Generation 4



(d) Generation 10

Figure 1: A snapshot of SCGA in a run on Problem Peaks1 - generation 1,2,4,10.

Algorithm 6 Species Specific Explosion

```
procedure SpeciesSpecificExplosion(G, X_s, K, SS)
    E_s ← ∅;
    WeightsEval(X_s, G);
    for ∀x ∈ X_s do
        if x is present in previous generation
        or SS = true then
            size ← x.weight × K × pop_size;
            E_s ← E_s ∪ Explode(x, size);
        end if
    end for
    return E_s;
end procedure

procedure Explode(x, size)
    Exploded_s ← ∅;
    for i from 1 to size do
        temp ← Copy of x;
        Mutate temp with step size x.delta;
        Exploded_s ← Exploded_s ∪ temp;
    end for
    return Exploded_s;
end procedure
```

3.5 Calculate Explosion Weights

Before an explosion, we need to determine the explosion weight for each species seed. The explosion weight is defined over $[0, 1]$. It is a scaling factor to determine the number of mutated copies that a species seed can create during the species-specific explosion process. In EASE, the rationale behind is to encourage a species to create more mutated copies if the species has less individuals in the current population. Hence the explosion weight of a species seed is derived from the ratio of individuals in the same species to the current population. The larger the ratio, the smaller is the explosion weight and vice versa. The algorithm is shown in Algorithm 7. Each explosion weight is normalized at the end so that the sum of all the explosion weights is limited to 1, in order to avoid the total number of the mutated copies of all species seeds exceeding the predefined value $K \times pop\_size$.

Algorithm 7 Calculate the explosion weight for each species seed

```
procedure WeightsEval(X_s, G)
    total ← 0;
    for ∀x ∈ X_s do
        x.weight ← population size - number of individuals
        in the same species in the current population;
        total ← total + x.weight;
    end for
    for ∀x ∈ X_s do
        x.weight ← x.weight/total;
    end for
end procedure
```

4. EXPERIMENT

We implemented EASE using Sun's Java programming language. Its development is based on the EC4 framework provided in Kenneth De Jong's book [8]. Experiments to compare the performance between EASE and SCGA were conducted on four benchmark functions as shown below:

- F1: Deb's 1st function [3]
- F2: Himmelblau function [1]
- F3: Six-hump Camel Back function [14]
- F4: Branin function [14]

Furthermore, the results of Fitness Sharing (FS) [5] and other algorithms as reported in [12, 11]: Roaming Agent-Based Collaborative Evolutionary Model (RACE) proposed in [11], Roaming Technique (RO) proposed in [12], Crowding Differential Evolution (CRDE) proposed in [18] and Adaptive Elitist Genetic Algorithm (AEGA) proposed in [9] are summarized and compared with the performance of EASE and SCGA. All the five benchmark functions have already been presented in [12]. Hence we just briefly describe each of them in this paper.

- Problem Peaks1 is defined on $[-100, 100]^2$ and has three peaks of different heights. The maximum height is 30, whereas the minimum height is 20.
- Problem Peaks2 is defined on $[-100, 100]^2$ and has ten peaks of different heights. The maximum height is 50, whereas the minimum height can reach to 10.
- Problem Peaks3 is defined on $[-10, 10]^2$ and has fifteen peaks of the same height. The heights of all peaks are 4.
- Problem Peaks4 is defined on $[-10, 10]^2$ and has the same peak locations as Problem Peaks3, but the heights are different. The heights of some peaks are changed to 5, 6 or 7.
- Problem Peaks5 is defined on $[-100, 100]^{10}$ and has four peaks of the same height. The heights of all peaks are 100.

4.1 Performance measurement

For multimodal optimization, there are several performance metrics proposed previously [17]. The focuses of this paper are on (1) the ability of the algorithms to locate the optima and (2) the accuracy of the optima found by the algorithms. Hence we use the peak ratio and the average minimum distance to the real optima [12] as the performance metrics.

- A peak is considered found when there exists an individual which is within 0.5 Euclidean distance to the peak in the last population. Thus the peak ratio is calculated using the following formula:

$$PeakRatio = \frac{Number\ of\ peaks\ found}{Total\ number\ of\ peaks}$$

- The average minimum distance to the real optima is calculated using the following formula:

$$D = \frac{\sum_{i=1}^{n} \min_{indiv \in pop} d(peak, indiv)}{n}$$

where n is the number of peaks, *indiv* denotes an individual and *pop* denotes a population of individuals.

Table 1: Common parameter setting of EASE and SCGA for different benchmarks

Benchmark	Population Size	Species Distance†
F1	100	0.1
F2	100	3
F3	100	1
F4	100	6
Peaks1	200	50
Peaks2	200	25
Peaks3	200	3
Peaks4	200	3
Peaks5	50	150

Table 2: Parameter setting of FS for different benchmarks

Benchmark	Population Size	Niche Radius	Scaling Factor
Peaks1	200	50	0.01
Peaks2	200	25	0.01
Peaks3	200	3	0.05
Peaks4	200	3	0.05
Peaks5	1000	150	1

In the following sections, all algorithms were run up to maximum 50000 fitness function evaluations. The above performance metrics are obtained by taking the average and standard deviation of 30 runs. The above setting is exactly the same as [12, 11].

4.2 Parameter settings

The parameter setting of EASE for all benchmarks is shown in Table 4. The common parameter setting of EASE and SCGA for all benchmarks is shown in Table 5. The common parameter setting of EASE and SCGA for different benchmarks is shown in Table 1. All the common parameter settings of EASE are exactly the same as SCGA for fair comparisons. The selection method of the species distance parameters is based on the suggestions in [10]. The parameter setting of FS for different benchmarks is shown in Table 2. For the other algorithms, the details for the settings can be referred to [12, 11].

4.3 Results

Table 3 shows the experimental results for the comparison of EASE and SCGA. It can be observed that EASE outperformed SCGA in all the benchmark functions. EASE did improve SCGA's performance no matter in the ability to locate optima or the accuracy of the optima found.

Table 6 shows the experimental results for the comparison of all algorithms tested. On the whole, EASE also showed its competitive results with other existing algorithms except Problem Peaks2. Looking at Problem Peaks2 more deeply as shown in Figure 2, it is defined on $[-100, 100]^2$ and has ten peaks of different heights. The maximum height of peaks is 50, whereas the minimum height of peaks can even reach down to 10. Such a large variation in peak height imposes a great challenge for algorithms to locate all the optima. Since we employ a fitness proportional method and truncation selection as parent selection method and survival selection method respectively in EASE, strong selection pressure has been imposed on the population. The individuals approaching the short peaks are often removed easily in each run. Thus it often cannot locate short peaks such that its peak ratio is always around 0.6, which is approximately equal to the number of tall peaks divided by the total number of peaks. Such situation can be corrected by using other selection methods, but the peak accuracy will be traded off. Nevertheless, once EASE locates a peak, it can always converge to the peak more precisely than others. Thus its measurement D can still be the best among the algorithms tested.

†Using the terms in [10], the species distance $(r_s) = \sigma_s/2$

Table 3: Experimental Results for the comparison of EASE and SCGA

Benchmark	Measurement	SCGA	EASE
F1	Mean of D	1.32E-03	**2.14E-10**
	StDev of D	9.87E-04	**6.71E-11**
	Min of D	1.20E-04	**1.22E-10**
	Median of D	1.03E-03	**2.01E-10**
	Mean of Peak Ratio	**1.000**	**1.000**
	StDev of Peak Ratio	**0.000**	**0.000**
F2	Mean of D	2.48E-01	**1.12E-06**
	StDev of D	1.27E-01	**3.07E-06**
	Min of D	5.72E-02	**5.44E-07**
	Median of D	2.11E-01	**5.44E-07**
	Mean of Peak Ratio	0.825	**1.000**
	StDev of Peak Ratio	0.187	**0.000**
F3	Mean of D	1.10E-02	**2.11E-06**
	StDev of D	1.49E-02	**9.36E-06**
	Min of D	1.28E-04	**3.91E-07**
	Median of D	5.65E-03	**4.02E-07**
	Mean of Peak Ratio	**1.000**	**1.000**
	StDev of Peak Ratio	**0.000**	**0.000**
F4	Mean of D	6.85E-01	**7.88E-07**
	StDev of D	5.75E-01	**1.07E-08**
	Min of D	6.99E-02	**7.61E-07**
	Median of D	5.28E-01	**7.89E-07**
	Mean of Peak Ratio	0.622	**1.000**
	StDev of Peak Ratio	0.243	**0.000**

Interestingly, the experimental results also reveal a critical observation about EASE. For some benchmarks tested, it shows a minority of relatively poor results which subsequently deteriorates the mean value of the measurement D. Such phenomenon can be exemplified by the experiment conducted on Problem Peaks1, in which the mean value of D differed from the median value of D by 10^5 magnitude order. It is mainly due to the stochastic property of EASE. Hence further efforts should be spent on stabilizing EASE for locating optima (global and local).

5. CONCLUSIONS

A new evolutionary algorithm for multimodal optimization called Evolutionary Algorithm with Species-specific Explosion(EASE) is proposed. EASE is an algorithm to remedy SCGA by exploding species seeds for locating optima.

EASE is divided into two stages: **Exploration Stage** and **Species-specific Stage**. EASE starts with the exploration stage. Once the stage switching condition is satisfied, it will be changed to species-specific stage. Throughout the two stages, a local operation: **Species-specific Explosion** is

applied so as to help each species to converge to its respective optimum.

The experimental results show that EASE improves SCGA for locating optima (global and local), in terms of peak ratio and accuracy. Comparing with the results reported in [12, 11], EASE also did a great job in all the five benchmark functions.

In the future, we will focus more on experiments with real-world multimodal optimization problems. High dimensional problems will also be considered. The concept **Species-specific Explosion** will be investigated to improve other evolutionary algorithms.

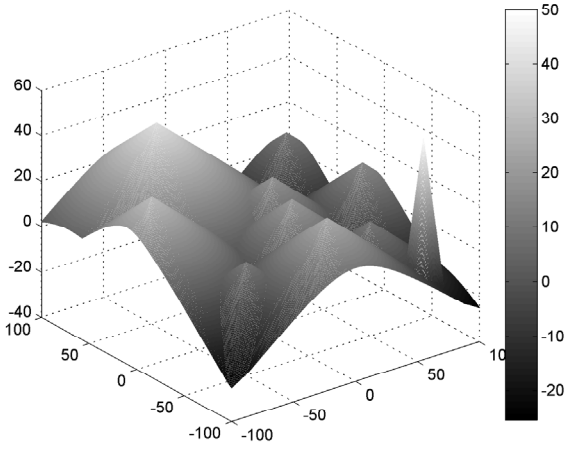


Figure 2: Problem Peaks2

6. ACKNOWLEDGMENTS

The authors are grateful to anonymous reviewers for their valuable comments. The authors would also like to thank Gulshan Singh for his source codes. This research is partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project Nos. 414107 and 414708).

7. REFERENCES

[1] D. Beasley, D. R. Bull, and R. R. Martin. A sequential niche technique for multimodal function optimization. *Evol. Comput.*, 1(2):101–125, 1993.

[2] M. Bessaou, A. Pétrowski, and P. Siarry. Island model cooperating with speciation for multimodal optimization. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 437–446, London, UK, 2000. Springer-Verlag.

[3] K. Deb and D. E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[5] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[6] J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA, 1992.

[7] K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Ann Arbor, MI, USA, 1975.

[8] K. A. D. Jong. *Evolutionary Computation. A Unified Approach*. MIT Press, Cambridge, MA, USA, 2006.

[9] K. S. Leung and Y. Liang. Adaptive elitist-population based genetic algorithm for multimodal function optimization. In *In GECCO 2003, LNCS 2723*, pages 1160–1171. Springer-Verlag, 2003.

[10] J. P. Li, M. E. Balazs, G. T. Parks, and P. J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evol. Comput.*, 10(3):207–234, 2002.

[11] R. I. Lung, C. Chira, and D. Dumitrescu. An agent-based collaborative evolutionary model for multimodal optimization. In *GECCO '08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 1969–1976, New York, NY, USA, 2008. ACM.

[12] R. I. Lung and D. Dumitrescu. A new evolutionary model for detecting multiple optima. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1296–1303, New York, NY, USA, 2007. ACM.

[13] R. Manner, S. W. Mahfoud, and S. W. Mahfoud. Crowding and preselection revisited. In *Parallel Problem Solving From Nature*, pages 27–36. North-Holland, 1992.

[14] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK, 1996.

[15] A. Petrowski. A clearing procedure as a niching method for genetic algorithms. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 798–803, Nagoya, Japan, May 1996.

[16] G. Singh and D. Kalyanmoy Deb. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1305–1312, New York, NY, USA, 2006. ACM.

[17] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real parameter optimization. Technical report, Nanyang Technological University, 2005.

[18] R. Thomsen. Multimodal optimization using crowding-based differential evolution. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1382–1389, June 2004.

*The value of 0 is approximated by the double-precision 64-bit IEEE 754 floating point standard

Table 4: Parameter setting of EASE for all benchmarks

Parameter	Setting
Mutation Type	Gaussian [8]
Mutation Formula in explosion	$NewValue = OldValue + 2 \times StepSize \times U$ where U is a normally distributed real value with mean 0.0 and standard deviation 1.0
K	0.4

Table 5: Common parameter setting of EASE, SCGA and FS for all benchmarks

Parameter	Setting
Population Initialization	Random
Generation Type	Overlapping [8]
Parent Selection	Fitness Proportional
Survival Selection	Truncation [8]
Representation	Sun's Java Double (double-precision 64-bit IEEE 754 floating point)
Mutation Type	Gaussian [8]
Mutation Formula	$NewValue = OldValue + 1.3 \times StepSize \times U$ where U is a normally distributed real value with mean 0.0 and standard deviation 1.0
Mutation Probability	0.2
Mutation Step Size	0.1
Crossover Type	Intermediate Recombination [10]
Crossover Formula	$Offspring = \frac{Parent_1 + Parent_2}{2}$
Crossover Probability	1
Random Seed	12345
Implementation	Sun's Java programming language

Table 6: Experimental Results for all algorithms tested

Benchmark	Measurement	FS	RACE	RO	AEGA	CRDE	SCGA	EASE
Peaks1	Mean of D	0.844	7.77E-07	0.246	0.915	4.68E-05	2.665	**3.87E-10**
	StDev of D	0.475	2.23E-07	0.779	1.272	1.44E-05	1.323	**2.06E-09**
	Min of D	0.257	2.6E-07	3.91E-05	0.058	2.71E-04	0.716	**0***
	Median of D	0.718	7.87E-07	6.81E-05	0.266	4.47E-04	2.667	**4.44E-15**
	Mean of Peak Ratio	0.500	**1**	0.966	0.8	**1**	0.400	**1.000**
	StDev of Peak Ratio	0.259	**0**	0.105	0.23	-	0.136	**0.000**
Peaks2	Mean of D	69.166	6.89	8.414	13.77	13.23	6.111	**4.393**
	StDev of D	13.087	4.62	4.000	6.419	**0.019**	1.307	3.212
	Min of D	48.112	**3.94E-06**	0.752	5.304	13.19	4.066	0.055
	Median of D	77.043	6.69	8.53	12.36	13.24	6.182	**3.839**
	Mean of Peak Ratio	0.103	**0.87**	0.75	0.46	0.7	0.117	0.603
	StDev of Peak Ratio	0.061	0.07	0.108	0.09	-	**0.038**	0.096
Peaks3	Mean of D	1.766	8.47E-06	3.52E-04	10.74	7.86E-03	0.392	**9.34E-07**
	StDev of D	0.353	**1.01E-06**	5.22E-05	1.391	9.92E-04	0.080	5.11E-06
	Min of D	1.236	7E-06	2.60E-04	8.459	5.85E-03	0.252	**3.77E-15**
	Median of D	1.698	8.51E-06	3.51E-04	10.88	8.07E-03	0.381	**3.95E-13**
	Mean of Peak Ratio	0.573	**1**	**1**	0.066	**1**	0.704	**1.000**
	StDev of Peak Ratio	0.069	**0**	-	-	-	0.092	**0.000**
Peaks4	Mean of D	2.108	0.02	1.92E-03	10.76	6.47E-03	0.525	**4.83E-06**
	StDev of D	0.172	0.07	3.20E-03	2.599	1.13E-03	0.121	**1.48E-05**
	Min of D	1.853	6.64E-06	3.11E-04	7.642	4.41E-03	0.347	**1.47E-14**
	Median of D	2.105	8.52E-06	4.41E-04	10.30	6.35E-03	0.504	**6.62E-13**
	Mean of Peak Ratio	0.409	0.99	**1**	0.066	**1**	0.602	**1.000**
	StDev of Peak Ratio	0.045	0.02	-	-	-	0.092	**0.000**
Peaks5	Mean of D	158.926	8.5E-04	8.565	18.32	59.21	136.969	**5.35E-04**
	StDev of D	26.889	**6.9E-05**	7.323	8.294	0.130	17.500	4.45E-04
	Min of D	102.276	6E-04	0.044	7.907	58.85	101.481	**3.19E-05**
	Median of D	157.070	8.5E-04	11.00	15.07	59.26	134.261	**3.99E-04**
	Mean of Peak Ratio	0.000	**1**	0.775	0.05	0.25	0.000	**1.000**
	StDev of Peak Ratio	0.000	**0**	0.184	0.105	-	**0.000**	**0.000**