A Non-revisiting Genetic Algorithm with Adaptive Mutation for Function Optimization

Saroj and Devraj

Deaprtment of Computer Science and Engg.
Guru Jambheshwar University of Science and Technology, Hisar
{ratnoo.saroj,devraj.kamboj}@gmail.com

Abstract. Genetic Algorithm (GA) is a robust and popular stochastic optimization algorithm for large and complex search spaces. The major disadvantages of Genetic Algorithms are premature convergence and revisits to individual solutions in the search space. In other words, Genetic algorithm is a revisiting algorithm that leads to duplicate function evaluations which is a clear waste of time and computational resources. In this paper, a non-revisiting genetic algorithm with adaptive mutation is proposed for the domain of function optimization. In this algorithm whenever a revisit occurs, the underlined search point is replaced with a mutated version of the best/random (chosen probabilistically) individual from the GA population. Moreover, the suggested approach is not using any extra memory resources to avoid revisits. To test the power of the method, the proposed non-revisiting algorithm is evaluated using nine benchmarks functions. The performance of the proposed genetic algorithm is superior as compared to simple genetic algorithm as confirmed by the experimental results.

Keywords: Function optimization, Genetic algorithm, Non-revisiting, Adaptive mutation.

1 Introduction

Developing new optimization techniques is an active area of research and Genetic Algorithm (GA) is a relatively new stochastic optimization algorithm pioneered by Holland [1]. A GA is capable of finding optimal solutions for complex problems in a wide spectrum of applications due to its global nature. A GA is an iterative procedure that maintains a population of structures that are candidate solutions to the specific problem under consideration. In each generation, each individual is evaluated using a fitness function that measures the quality of solution provided by an individual. Further, genetic operators such as reproduction, crossover and mutation are applied to introduce the diversity in the candidate solutions. In fact, a GA mimics the natural principle of survival of the fittest. A fitness proportionate selection and GA operators ensures the better and better fit solutions to emerge in successive generations [2][3][4]. However, GAs are not without limitations. Two of the main problems are- 1) premature convergence i.e. many a times a GA converges to some local optimal

N. Meghanathan et al. (Eds.): CCSIT 2012, Part II, LNICST 85, pp. 288–297, 2012.

solution. 2) Redundant function evaluations. A simple genetic algorithm do not memorizes the search points or solutions to the problem that it visits in its life time and it revisits lots of search points generating duplicate solutions resulting into redundant fitness computations. Here, a revisit to a search position x is defined as a re-evaluation of a function of x which has been evaluated before. The problem of revisit is all the more severe towards the end of a GA run. In many domains, the fitness evaluation is computationally very expensive and lots of time is wasted in revisiting the parts of the search space and duplicate function evaluations.

In this paper, we propose an improved GA with adaptive mutation operator to avoid revisits and redundant fitness evaluations to a large extent. This GA has the elitist approach and retains the best individual in every new population. A look up for revisits is made only in the current population along with the population of previous generation. If any individual produced is found duplicate, it is replaced probabilistically with a mutated version of the best individual or of a random individual. The mutation operator is adaptive in the sense that its power of exploration decreases and power of exploitation increases with the number of generations.

The proposed approach demonstrates that the duplicate removal introduces a powerful diversity preservation mechanism which not only results in better final-population solutions but also avoids premature convergence. The results are presented for nine benchmark functions and illustrate the effectiveness of duplicate removal through adaptive mutation. The results are directly compared to a simple GA which is not removing duplicate solutions generated in its successive generations.

The rest of the paper is organized as below. Section II describes the related work. The proposed non revisiting Genetic algorithm which removes duplicate individuals through adaptive mutation is given in section III. Experimental design and results are enlisted in section IV. Section V concludes the papers and points to the future scope of this work.

2 Related Work

A number of implementations of genetic algorithms in a wide spectrum of applications have been reported in the literature [1][2][3][4][5]. Since the inception of genetic algorithms, several advanced GA approaches came up improving the efficiency and efficacy of the results achieved in the domain of function optimization along with various other domains. Many of these approaches addressed the problem of premature convergence and revisiting behavior of genetic algorithms. One of these advance approaches is devising new GA operators that can be adapted to produce suitable individuals by considering the state of the current population with respect to global optimal solution[6][7][8][9]. Srinivas and Patnaik (1994) employed a self adapted GA that achieved global optima more often than a simple GA for several multimodal functions [8]. In their work, they have suggested use of adaptive probabilities of crossover and mutation to maintain adequate diversity to avoid premature convergence. Another significant contribution came from Herrera and Lozano (2000) who suggested heterogeneous gradual distributed real coded genetic

algorithms (RCGAs) for function optimization domain. They devised new fuzzy connective based crossovers with varying degree of exploration and exploitation which proved very effective to avoid premature convergence and achieved one of the best reported results for several benchmark problems of function optimization [9]. Friedrich et al. (2007) have made an important contribution by analyzing the influence of simple diversity mechanisms used in selection procedures on the runtime behaviour [14].

The other important problem with GAs that has been in the recent focus of the GA research community is redundant function evaluations and revisits. Mauldin [10] was among the first ones who enhanced the performance of a genetic algorithm by eliminating duplicate genotypes during a GA run. Mauldin used a uniqueness operator that allowed a new child $\boldsymbol{x}$ to be inserted into the population if $\boldsymbol{x}$ was greater than a Hamming-distance threshold from all existing population genotypes. Davis [4] also showed that a binary coded GA for a comparable number of child evaluations that removes duplicates in the population has superior performance. Eshelman and Schaffer [11] reconfirmed this observation by using selection based innovations and new GA operators that prevented revisits. The problem of duplicate function evaluations has also been addressed by providing GA a short/long term memory i.e. the GA stores all the search points visited and their corresponding fitness into some data structure. In such approaches every time a new search point is produced by GA, before actually computing its fitness, the memory of GA is looked into and if this search point exists, its fitness is not recomputed. If the new solution is not in the memory, its fitness is computed and appended to the memory. Binary search trees, Binary partition tress, heap, hash tables and cache memory have been used to provide the supplement memory to GA. Such strategies have resulted in performance enhancement of GAs by eliminating revisits partially or completely [12][13][14][15]. Recently, Yuen and Chow [16] used a novel binary space partitioning tree to eliminate the duplicate individuals. Saroj et al. used a heap structure to avoid the redundant fitness evaluations in domain of rule mining. Their approach proved to be effective for large datasets. [17]. Though, all these duplicate removal method reduce the run time of a GA, they require huge data structures and a significant amount of time is spent for memory look ups. It is not uncommon for a GA to run for thousands of generations with a population of hundreds of individuals. If we assume a GA with 100 individuals and 5000 generations, we shall need a data structure that can store 250000 problem solutions and that is when we assume half the individuals produced are duplicates. The GA shall require 500000 look ups to avoid redundant fitness computation. GAs is already considered slow as compared to other optimization techniques and these approaches further slow down GA's performance. Clearly this method is successful only in the domains where fitness computations time is significantly larger than the memory look ups time and not suitable at all for domain of function optimization where fitness evaluation is relatively less expensive. Therefore, in this paper, we have adopted a genetic algorithm approach that maintains the diversity through adaptive mutation, avoids revisits to a large extent and that is without any additional memory overheads.

3 Proposed Non-revisiting GA with Adaptive Mutation

Non-revisiting algorithm is the one which do not visit the search points already visited. The improved GA with non-revisiting algorithm and adaptive mutation has to perform some extra steps than a simple GA. These steps are used to found the duplicate individuals. If any duplicate individual is found then it is mutated and reinserted in the current population. The duplicates are looked with respect to current and the previous generation only. There is a special condition that the best individual is preserved and not mutated. The flow chart and algorithm for the proposed GA is given in Fig. 1 and Fig 2 respectively.

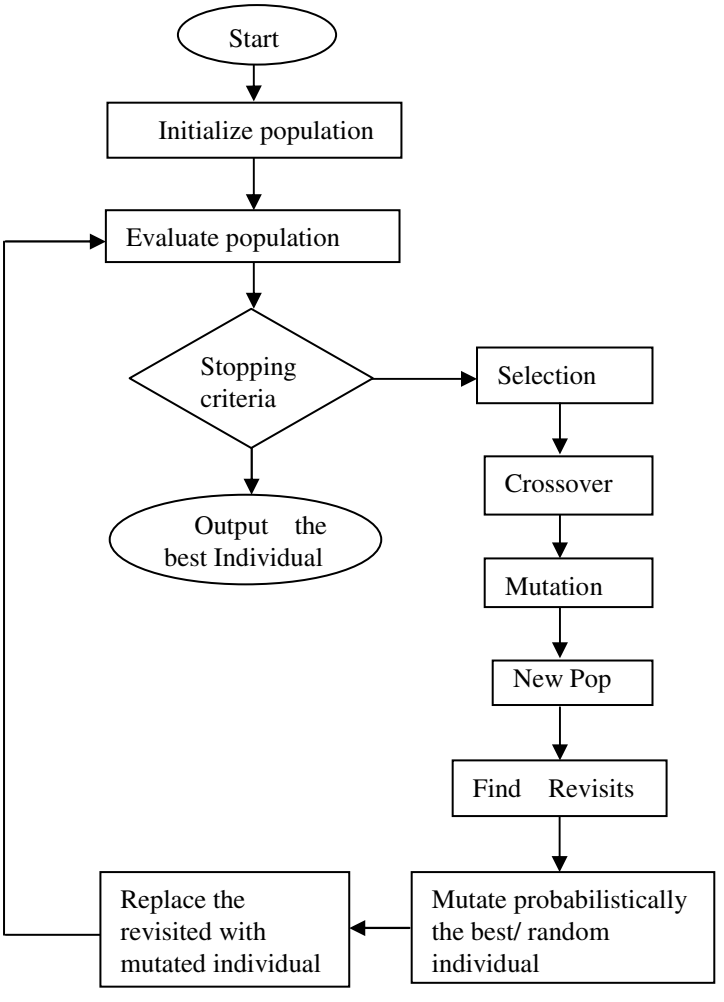

Fig. 1. Step by step procedure for Non-revisiting GA with adaptive mutation

The mutation applied is Gaussian adaptive mutation. The Gaussian function generates a random number around zero mean. The formula for mutation is as follows.

$$\text{mscale} = \text{mscale} - \text{mshrink} * \text{mscale} * (\text{current\_generation} / \text{total\_generations}) \quad (1)$$

$$x(i) = \text{best\_x} \pm (\text{gaussian\_rand} * \text{mscale}) \quad (2)$$

The amount of mutation is controlled by the two parameters mscale and mshrink. Here, mscale represents the variance for mutation for the first generation and mshrink represents the amount of shrink in mutation in successive generations. The mutation scale decreases as the number of generation increase. It is clear from the above formulae that such kind of mutation is exploratory during the initial runs and exploitative towards the final runs of the GA. We have kept the mscale as 1.0 and mshrink equals to 0.75.

```
1. Begin
2. Initial condition: Initial population:- old-pop=[],  new-pop[], Stopping Flag:-
    SF=0, Best Fit=0, Visiting Flag:- VF[]=0
3. Initialize the population in old-pop
4. Evaluate the fitness of old-pop and calculate the overall best individual up to
     the current generation.
5. If(stopping criteria=yes)
    5.1  SF=1   //set stopping flag
    5.2  Output the best individual
    5.3   Stop
       Else
    5.4 Copy the best individual into new-pop
    5.5 Perform a GA step by applying GA operators i.e. selection, crossover and
         mutation.
    5.6 Maintain the old population and store the newly generated individuals in
         the new pop.
    5.7 For (i=1 to pop-size)
              check revisits within the new-pop  and   with respect to old-pop
    5.8  If  revisit && not best_individual)
    5.9                VF[i]=1
    5.10  For(i=1 to pop-size)
    5.11  If  VF[i]==1
                   New-pop[i] =mutated (new-Pop[i])
    5.12 old-pop=new-pop
Go to step 4
```

Fig. 2. Algorithm for the proposed GA with no-revisit

The proposed non-revisiting GA with adaptive mutation has three key strengths.

1. It automatically assures maintenance of diversity and prevents premature convergence. Most of the individuals in a GA population are guaranteed to be different. By nature of the proposed GA, it is impossible for a population to consist of one kind of individuals only.
2. It might not completely eliminate the revisits. However it doesn't require large data structure to store the individuals to do a look up for duplicates and only uses the previous and current populations which are anyway available.
3. It probabilistically takes the advantage of the best individuals and converges faster without suffering problem of convergence.

4 Experimental Results

We have implemented the proposed non-revisiting GA with adaptive mutation on nine Benchmarks functions in four dimensions and the set of test functions is given below.

4.1 Test Function Set

1. Rastrigin's function
2. Sphere function
3. Generalized Rosenbrock function
4. Generalized Rastrigin function
5. Griewank's function
6. Ackley function
7. Rotated Griewank's function
8. Rotated Weierstrass's function
9. Branin function

All these function are shown in detail in the Appendix I. The first four functions are unimodal functions; the next five are multimodal functions designed with a considerable amount of local minima. The eighth and ninth functions are rotated multimodal functions. The improved GA is implemented in MATLAB. A comparison is made between a simple GA and the proposed GA on the basis of mean and the best fitness over the generations. The best fitness is the minimum score of the population. Both the GA's stop when there is no improvement in the best score over last fifty generations. The population size is kept at 20 for all the functions and, the crossover and mutation rates are equal to 0.6 and 0.01 respectively. The normal mutation rate is kept low as adaptive mutation to remove duplicates is also applied. The best individual or a random individual is mutated with equally likely (probability = 0.5) to replace the revisited points in the new population. We have used real encoding, proportion scaling, remainder stochastic selection, one point crossover and Gaussian mutation. These results are averaged over 20 runs of the GA. A comparison on the basis of mean and best fitness of the final population of the proposed GA and simple GA for nine benchmarks functions is shown Table 1.

Table 1. Best and mean fitness of the final populations of SGA and Non-revisiting GA with adaptive mutation

Benchmarks Function	Best Fitness		Mean Fitness	
	SGA	NRGA	SGA	NRGA
Rastrigin's function	2.561	1.023	21.68	19.81
Sphere function	0.004	0.002	1.474	1.102
Generalized Rastrigins function	2.461	0.024	18.47	15.71
Griewank's function	0.0020	0.0021	1.107	1.062
Generalized Rosenbrock function	2.834	0.091	21.24	19.24
Ackley function	3.324	0.009	22.42	18.96
Rotated Griewank's function	0.009	0.001	1.104	1.014
Rotated Weierstrass's function	3.784	0.092	1.936	1.526
Branin function	3.187	0.335	16.16	14.01

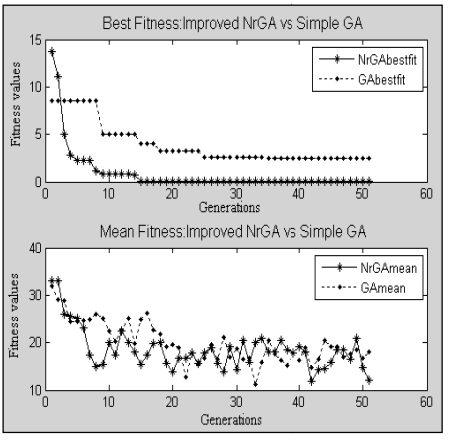

Fig. 3. Performance comparison of NRGA and SGA for Generalized Rastrigin's function

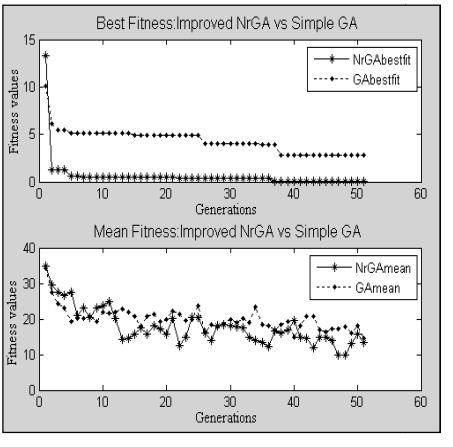

Fig. 4. Performance comparison of NRGA and SGA for Generalized Rosenbrocks's function

Though, our approach performs better for all the nine benchmark functions, Fig. 3 to Fig. 6 show the performance comparison of non-revisiting GA with simple GA for Generalized Rastrigins, Rosenbrock, Ackley and Rotated Griewank functions where NRGA approach dominates significantly. It is quite clear from the results that the performance of the non-revisiting GA with adaptive mutation is better than the simple GA.

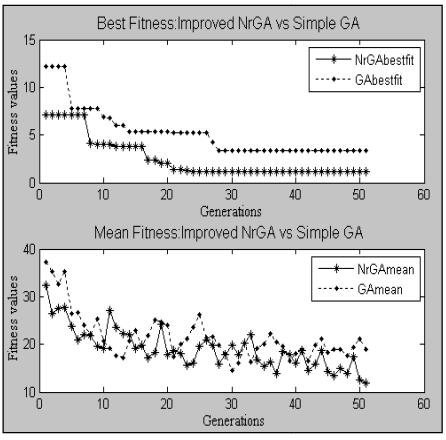

Fig. 5. Performance comparison of NRGA and SGA for Ackley's function

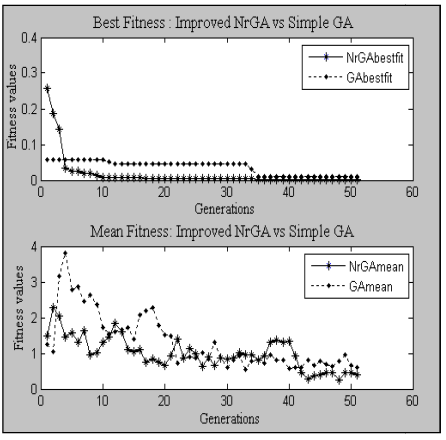

Fig. 6. Performance comparison of NGGA and SGA for Generalized Griewank's

5 Conclusion

In this paper, a novel non-revisiting GA with adaptive mutation is proposed and tested in the domain of function optimization. Though new improved GA may not completely eliminate the revisits and redundant function evaluation, it guarantees enough diversity to avoid the problem of premature convergence. The envisaged approach achieves better accuracy without much overheads of searching time for duplicates individuals and large data structures to serve as the long term memory for a GA as suggested in several earlier approaches [12][13][14][15].

The mechanism of a probabilistic adaptive mutation provides the much required balance between exploration and exploitation along with faster convergence to the optimal. It is exploratory in the initial runs of GA and exploitative towards the final runs of GA. More the number of generations of the GA, smaller will be the change in the new individual that replaces the revisited search point. The experimental results are very encouraging and show that the improved GA is clearly superior to its conventional counterpart. The adaptation of the current approach is underway for the domain of rule mining in the field of knowledge discovery.

References

[1] Holland, J.H.: Adaptation in Natural and Artificial Systems. MI Univ. Michigan Press, Ann Arbor (1975)
[2] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, New York (1989)
[3] Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, Heidelberg (1999)
[4] Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)

[5] De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems, PhD Thesis, University of Michigan, Ann Arbor, MI, USA (1975)
[6] Srinivasa, K.G., Venugopal, K.R., Patnaik, L.M.: A Self Adaptive Migration Model Genetic Algorithm for Data Mining Applications. Information Sciences 177, 4295–4313 (2007)
[7] Ono, I., Kita, H., Kobayashi, S.: A Robust Real-Coded Genetic Algorithm using Unimodal Normal Distribution Crossover Augmented by Uniform Crossover: Effects of Self-Adaptation of Crossover Probabilities. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 1, pp. 496–503. Morgan Kaufmann, Orlando (1999)
[8] Srinivas, M., Patnaik, L.M.: Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. IEEE Transactions on Systems, Man and Cybernetics 24, 656–667 (1994)
[9] Herrera, F., Lozano, M.: Gradual Distributed Real-Coded Genetic Algorithms. IEEE Transactions on Evolutionary Computation 4, 43–63 (2000)
[10] Mauldin, M.L.: Maintaining Diversity in Genetic Search. In: Proceeding National Conference on Artificial Intelligence, Austin, pp. 247–250 (1984)
[11] Eshelman, L., Schaffer, J.: Preventing Premature Convergence in Genetic Algorithms by Preventing Incest. In: Proceedings of the Fourth International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo (1991)
[12] Povinelli, R.J., Feng, X.: Improving Genetic Algorithms Performance by Hashing Fitness Values. In: Proceedings Artificial Neural Network, England, pp. 399–404 (1999)
[13] Kratica, J.: Improving the Performances of Genetic Algorithm by Caching. Computer Artificial Intelligence 18, 271–283 (1999)
[14] Friedrich, T., Hebbinghaus, N., Neumann, F.: Rigorous Analysis of Simple Diversity Mechanisms. In: Proc. Genetic Evolutionary Computation Conference, pp. 1219–1225. ACM Press, London (2007)
[15] Ronald, S.: Duplicate Genotypes in a Genetic Algorithm. In: Proc. IEEE Int. Conf. Evolutionary Computation, Anchorage, Alaska, pp. 793–798 (1998)
[16] Yuen, S.Y., Chow, C.K.: A Non-revisiting Genetic Algorithm. In: Proc. IEEE Congress on Evolutionary Computation, Singapore, pp. 4583–4590 (2007)
[17] Saroj, Kapila, Kumar, D., Kanika: A Genetic Algorithm with Entropy Based Probabilistic Initialization and Memory for Automated Rule Mining. In: Meghanathan, N., Kaushik, B.K., Nagamalai, D. (eds.) CCSIT 2011, Part I. CCIS, vol. 131, pp. 604–613. Springer, Heidelberg (2011)

Appendix I: Benchmarks Function

1. Rastrigin's function: $f1(x) = 10x+\sum_{i=1}^{D}[x^2\text{-}10\cos(2\pi x)+10]$
   Where $x \in [-5.12, 5.12]^D$
   Min $f1(x) = f1([0, 0\ldots.0]) = 0$
2. Sphere function [14]: $f2(x) = \sum_{i=1}^{D} x^2$
   Where $x \in [-5.12, 5.12]^D$
   Min $f2(x) = f2([0, 0\ldots.0]) = 0$
3. Generalized Rosenbrock function: $f3(x) = \sum_{i=1}^{D-1}[100(x_{i+1}\text{-}x_i^2)^2 + (x_i\text{-}1)^2]$
   Where $x \in [-5.12, 5.12]^D$
   Min $f3(x) = f3([1, 1\ldots.1]) = 0$
4. Generalized Rastrigin function: $f4(x) = \sum_{i=1}^{D}[x^2\text{-}10\cos(2\pi x)+10]$
   Where $x \in [-5.12, 5.12]^D$
   Min $f4(x) = f4([0, 0\ldots 0]) = 0$
5. Griewank's function: $f5(x) = \frac{1}{4000}\sum_{i=1}^{D} x^2\text{-}\prod_{i=1}^{D} cos\frac{xi}{\sqrt{i}} + 1$
   Where $f5(x) \in [-5.12, 5.12]^D$
   Min $f5(x) = f5([0, 0\ldots 0])$
6. Ackley function: $f6(x) = -20\exp(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2})\text{-}\exp(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i) + 20 + e$
   where $x \in [-5.12, 5.12]^D$
   Min $f6(x) = f6([0, 0\ldots 0]) = 0$
7. Rotated Griewank's function: $f7(x) = \frac{1}{4000}\sum_{i=1}^{D} z^2\text{-}\prod_{i=1}^{D} cos\frac{zi}{\sqrt{i}} + 1$
   Where $z=xM$, $f7(x) \in [-5.12, 5.12]^D$
   Min $f7(x) = f7([0, 0\ldots 0])$
8. Rotated Weierstrass's function [14]:
   $f8(x) = \begin{bmatrix} 1 & \cdots & D \\ \vdots & \ddots & \vdots \\ D2-D & \cdots & D2 \end{bmatrix}$ //$D2=D^2$
9. Branin function: $f9(x) = (x_2\text{-}\frac{5}{4\pi^2}x_1^2+\frac{5}{\pi}x_1\text{-}6)^2+10(1-\frac{1}{8\pi})\cos x_1+10$
   Where $x \in [-5.12, 5.12]$
   Min $f9(x) = f9([-3.142, 12.275]) =$
   $f9([3.142, 2.275])$