Search Direction Made Evolution Strategies Faster

Guangming Lin[1], Xin Lu[1], and Lishan Kang[2]

[1] Shenzhen Insititute of Information Technology, Shenzhen, China
[2] School of Computer Science, China University Geoscience, Wuhan, China

Abstract. Genetic operators are primarily search operators in Evolution Strategies (ES). In fact, there are two important issues in the evolution process of the genetic search: exploration and exploitation. The analysis of the impact of the genetic operators in ES shows that the Classical Evolution Strategies (CES) relies on Gaussian mutation, whereas Fast Evolution Strategies (FES) selects Cauchy distribution as the primary mutation operator. With the analysis of the basic genetic operators of ES as well as their performances on a number of benchmark problems, this paper proposes an Improved Fast ES (IFES) which applies the search direction of global optimization into mutation operation to guide evolution process convergence, thus making the process quicker.

Extensive empirical studies have been carried out to evaluate the performances of IFES, FES and CES. The experimental results obtained from four widely used test functions show that IFES outperforms both FES and CES. It is therefore concluded that it is important to strike a balance between exploration and exploitation.

Keywords: Search Direction Mutations, Evolution Strategies, Exploration and Exploitation.

1 Introduction

Evolutionary Algorithms (EAs) are a powerful stochastic search and optimization technique based on the mechanism of natural evolution. It is applicable to a wide range of problems for which little prior knowledge is available. Genetic operators play an important role in EAs. In general, an abstract task which is to be accomplished by EAs can be perceived as a search through a space of potential solution. Since usually we are following "the best" solution, we can view this task as an optimization process. For large search spaces, the methods of EAs are more sufficient than the classical exhaustive methods; they are stochastic algorithms whose search methods model some natural phenomena: genetic inheritance and Darwinian strife for survival [9]. The best known techniques in the class of EAs are Genetic Algorithms (GAs), Evolution Strategies (ESs), Evolutionary Programming (EP), and Genetic Programming (GP).

The Classical ES and Classical EP fist use a Gaussian mutation to generate new offspring and then use the deterministic selection to test them. There has been lots of work on different selection schemes for ES [1]. However work on mutations has been concentrated on self adaptation [2, 3] rather than new operators. Gaussian mutations

Z. Cai et al. (Eds.): ISICA 2009, CCIS 51, pp. 146–155, 2009.

seemed to be the only choice [2, 3]. Recently, Cauchy mutation has been proposed as a very promising search operator due to its higher probability of making long jumps [4, 5, 6]. However in [4, 5, 7] a fast EP and a fast ES based on Cauchy mutation were proposed.

It seems that there are two important issues in the evolution process of the genetic search: exploration and exploitation. Exploration is the creation of population diversity by exploring the search space; exploitation is the reduction of the diversity by focusing on the individuals of higher fitness, in other words, exploiting the fitness information (or knowledge) represented within the population. These factors are strongly related: an increase in the exploitation decreases the diversity of the population or the exploration, and vice versa. In other words, strong exploitation "supports" the premature convergence of the genetic search, but has excellent ability to tune solutions when they are near optima; a weak exploitation can make the genetic search ineffective, because exploration is lack of the power to improve the solution quality.

Based on the optimization theory, there are two important factors: search direction and search step size to affect the performance of the search algorithms. When the search points are far away from the global optimum in the initial search stages, increasing the search step size will increase the probability of escaping from a local optimum, and if the search direction is correct, it also has higher probability to reach the global optimum. On the other hand, with the progress of search, the current search points are likely to move closer and closer towards a global optimum. So it is necessary to restrict the step size in the later stages. However, it is hard to know in advance whether the search points are far from the global optimum. Unfortunately, the probability that a randomly generated initial population is very close to a global optimum is quite small in practice. It certainly worth enlarging the search step size in the early stages when we use EAs. In the final stages, the population of EAs tends to converge, and the step size tends to be reduced.

The rest of this paper is organized as follows. Section 2 formulates the global function optimization problem considered in this paper and describes the implementation of CES and FES. Section 3 analyzes the impact of the genetic operators on the exploration-exploitation tradeoff. Section 4 describes the implementation of VFES. Section 5 presents some discusses the experimental results. Finally, section 6 concludes with some discussions.

2 Function Optimization by Classical and Fast Evolution Strategies

A global minimization problem can be formalized as a pair (S, f), where $S \subseteq R^n$ is a bounded set on R^n and f: $S \mapsto R$ is an n-dimensional real-valued function. The problem is to find a point $x^{min} \in S$ such that $f(x^{min})$ is a global minimum on S. More specifically, it is required to find an $x^{min} \in S$ such that $\forall x \in S : f(x_{min}) \leq f(x)$, Where f does not need to be continuous but, it must be bounded. This paper only considers unconstrained function optimization.

2.1 Classical Evolution Strategies

According to the description by Bäck and Schwefel [7], in this study (μ,λ)-CES is implemented as follows:

1. Generate the initial population of μ individuals, and set k=1. Each individual is taken as a pair of real-valued vectors, $(x_i,\eta_i), \forall i \in \{1,2,...,\mu\}$, where x_i's are variables and η_i's are standard deviations for Gaussian mutations (also known as strategy parameters in self-adaptive evolutionary algorithms).
2. Evaluate the fitness score for each individual $(x_i,\eta_i), \forall i \in \{1,2,...,\mu\}$, of the population based on the objective function, $f(x_i)$.
3. Each parent $(x_i,\eta_i), \forall i \in \{1,2,...,\mu\}$, creates λ/μ offspring on average, so that a totalλ offspring are generated: for i=1,…, μ , j=1,…,n, and k=1,…, λ ,

$$x_k\text{'(j)} = x_i\text{(j)} + \eta_i\text{(j) N (0,1)} \tag{2.1}$$

$$\eta_k\text{'(j)} = \eta_i\text{ (j)exp(} \tau\text{'N(0,1)+ } \tau\text{ N}_j\text{ (0,1))} \tag{2.2}$$

   Where x_i(j), x_k'(j), η_i(j) and η_k'(j) denote the j-th component of the vectors x_i, x_k', η_i and η_k', respectively. N(0,1) denote a normally distributed one-dimensional random number with mean 0 and standard deviation 1. N_j(0,1) indicates that a new random number is generated for each value of j. The τ and τ' are commonly set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$ [7,8].
4. Calculate the fitness of each offspring $\left(x_i', \eta_i'\right), \forall i \in \{1,2,..., \lambda\}$.
5. Sort offspring $\left(x_i', \eta_i'\right), \forall i \in \{1,2,..., \lambda\}$ in a non-descending order according to their fitness values, and selection the μ best offspring out of λ to be parents of the next generation.
6. Stop if the stopping criterion is satisfied; otherwise, k=k+1 and go to Step 3.

2.2 Fast Evolution Strategies

The one-dimensional Cauchy density function centered at the origin is defined by:

$$f_t(x) = \frac{1}{\pi}\frac{t}{t^2 + x}, \qquad -\infty < x < \infty \tag{2.3}$$

where t>0 is a scale parameter. The corresponding distribution function is

$$F_t(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{x}{t}\right).$$

The FES studied in this paper is exactly the same as CES described in last section except for Eq. (2.1) which is replace by the following [6]:

$$x_i'(j) = x_i(j) + \eta_i(j)\ \sigma_j \tag{2.4}$$

Where σ_j is a Cauchy random variable with the scale parameter t=1, and is generated anew for each value of j. It is worth indicating that we leave Eq.(2.2) unchanged in FES in order to keep our modification of CES to a minimum. It is also easy to investigate the impact of the Cauchy mutation on ES when other parameters are kept the same.

3 Analysis the Impact of the Genetic Operators in ES

Evolution Algorithms (EAs) is a class of direct probabilistic search algorithm based on the model of organic evolution. In general, all EAs are characterized by the fact that they work on a population of individuals, each of which represents a search point in the space of potential solutions to a given optimization problem. The population undergoes subsequent modification steps by means of randomized genetic operators that are intended to model recombination (or crossover in GAs), mutation, and selection. The allocation of responsibilities to the genetic operators is relatively well understood: Mutation introduces innovations into the population, recombination changes the context of already available, useful information, and selection directs the search towards promising regions of the search space. Acting together, mutation and recombination *explore* the search space while selection *exploits* the information represented within the population. Next we will analyze the impact of these different genetic operators or various designs of any genetic operator on the balance between exploration and exploitation.

3.1 The Impact of Selection Operators

The selection operator uses the fitness information to favor individuals of higher quality to transfer their information to the next generation of the evolution process. It provides a mechanism to affect this balance towards exploitation—by increasing emphasis on the better individuals; or towards exploration—by providing similar chances to survive even for worse individuals. Informally, the term *selective pressure* is widely used to characterize the strong (high selective pressure) or the weak (small selective pressure) emphasis of selection on the best individuals. Selection operators guide individuals towards the better solutions according to the better search directions.

3.2 The Impact of Crossover Operators

The crossover operator usually destroys the links between parents and children. In other words, this operator focuses more on exploration but less on exploitation. When the population is almost with the same pattern, the crossover operator is lack of the power to improve the solution quality. The different crossover operators are analyzed in terms of *crossover point.* Under certain conditions a multi-point crossover is better

in exploration than a one-point one, because increasing the number of crossover point will increase crossover's search step size.

3.3 The Impact of Mutation Operators

Like crossover, the term *step size* is used to characterize the different mutation operators. This is probably due to the fact that the large search step size mutation operator can more easily generate offspring that are quite distant from their parents occasionally, and thus more easily escape from local minima as compared with the small search step size mutation operator. On the other hand, the small search step size mutation operator could lead to finer local search.

4 An Improved Fast Evolution Strategies

Generally, Cauchy mutation performs better when the current search point is away from the global minimum, while Gaussian mutation is better at finding a local optimum in a given region. It would be ideal if Cauchy mutation is used when search points are far away from the global optimum and Gaussian mutation is adopted when search points are in the neighborhood of the global optimum. Unfortunately, the global optimum is usually unknown in practice, making the ideal switch from Cauchy to Gaussian mutation very difficult. Self-adaptive Gaussian mutation [7,2,9,15,16] is an excellent technique to partially address the problem. That is, the evolutionary algorithm itself will learn when to "switch" from one step size to another. However, there is room for further improvement to self-adaptive algorithms like CES or even FES.

In this section, we propose an improved FES (IFES) base on *mixing* (rather than switching) different mutation operators. The idea is to mix different search biases of Cauchy and Gaussian mutation. The implementation of IFES is very simple. It differs from FES and CES only in Step 3 of the algorithm described in Section 2.1. Instead of using Eq. (2.1) (for CES) or Eq. (2.4) (for FES) alone, IFES generates two offspring from each parent, one by Cauchy mutation and the other by Gaussian. The better one is then chosen as the offspring. The rest of the algorithm is exactly the same as FES and CES. Chellapilla [10] has presented some more results on comparing different mutation operators in EP.

We define a Boolean function "*better*" as:

$$better(X_1, X_2) = \begin{cases} W(X_1) < W(X_2) & \text{TRUE} \\ W(X_1) > W(X_2) & \text{FALSE} \\ (W(X_1) = W(X_2)) \wedge (f(X_1) \le f(X_2)) & \text{TRUE} \\ (W(X_1) = W(X_2)) \wedge (f(X_1) > f(X_2)) & \text{FALSE} \end{cases}$$

If *better* (X_1, X_2) is TRUE, this means that the individual X_1 is "better" than the individual X_2.

The new algorithm can now be described as follows:

Begin

initialize $P = \{Z_1, Z_2, \ldots, Z_N\}$; $Z_i \in D^*$;

$t := 0$;

$Z_{best} = \arg \underset{1 \le i \le N}{Min} f(Z_i)$;

$Z_{worst} = \arg \underset{1 \le i \le N}{Max} f(Z_i)$;

while not abs ($F(Z_{best}) - F(Z_{worst})) \le \varepsilon$ **do**

select randomly M points $Z_1', Z_2', \ldots, Z_M'$ from P to form the subspace V;

select s points *randomly* $Z_1^*, Z_2^* \ldots Z_s^*$ from V;

for i=1,…s **do**

for j=1,…p+q **do**

$Z_{Gi}^*(j) := Z_i^*(j) + \sigma_i(j) N_j(0, 1)$

$Z_{Ci}^*(j) := Z_i^*(j) + \sigma_i(j) C_j(1)$

$\sigma_i(j) := \sigma_i(j) \exp(\tau' N(0,1) + \tau' N_j(0,1))$

endfor

if *better*(Z_{Gi}^*, Z_{Ci}^*)

then $Z_i^{'*} := Z_{Gi}^*$ else $Z_i^{'*} := Z_{Ci}^*$;

endfor

$Z' = \arg \underset{1 \le i \le N}{Min} f(Z_i)$;

if *better* (Z', Z_{worst}) **then** $Z_{worst} := Z'$;

$t := t + 1$;

$Z_{best} = \arg \underset{1 \le i \le N}{Min} f(Z_i)$;

$Z_{worst} = \arg \underset{1 \le i \le N}{Max} f(Z_i)$;

if abs $(f(Z_{best}) - f(Z_{worst})) \le \eta$ **.and.** $M \ge 3$ **then**

$M := M - 1$;

endwhile

output t , Z_{best} , $f(Z_{best})$;

end

Where $Z_{Gi}^*(j)$, $Z_{Ci}^*(j)$ and $\sigma_i(j)$ denote the j-th component of the vectors Z_{Gi}^*, Z_{Ci}^* and σ_i , respectively. N(0,1) denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0, 1)$ indicates that the Gaussian random number is generated anew for each value of j. $C_j(1)$ denotes a Cauchy distributed one-dimensional random number with *t=1*.

The factors τ and τ' have commonly set to $\left(\sqrt{2\sqrt{(p+q)}}\right)^{-1}$ and $\left(\sqrt{2(p+q)}\right)^{-1}$.

The new algorithm has the two important features:

(1) This algorithm is an ergodicity search. During the random search of the subspace, we employ a "non-convex combination" approach, that is, the coefficients a_i of $Z'=\sum_{i=1}^{m} a_i Z_i$ are random numbers in the interval [-0.5, 1.5]. This ensures a non-zero probability that any point in the solution space is searched. This ergodicity of the algorithm ensures that the optimum is not ignored.

(2) The monotonic fitness decrease of the population (when the minimum is required). Each iteration ($t \rightarrow t+1$) of the algorithm discards only the individual having the worst fitness in the population. This ensures a monotonically decreasing trend of the values of objective function of the population, which ensures that each individual of the population will reach the optimum.

5 Experimental Studies

In order to further explain the impact, we have studied some functions by our research on ESs.

Also we drew some conclusions from the results of the comparison of the different genetic operators. The following sets of functions were studied:

(Corridor) $F_1(x) = -\sum_{i=1}^{n} x_i, \quad n = 3$

Constraints:

$$G_j(x) = \begin{cases} -x_j + 100 \geq 0, & j = 1(1)n \\ x_{j-n+1} - \dfrac{1}{j-n}\sum_{i=1}^{j-n} x_i + \sqrt{\dfrac{j-n+1}{j-n}} \geq 0, & n+1 \leq j \leq 2n-1 \\ -x_{j-2n+2} + \dfrac{1}{j-2n+1}\sum_{i=1}^{j-2n+1} x_i + \sqrt{\dfrac{j-2n+2}{j-2n+1}} \geq 0, & 2n \leq j \leq 3n-2 \end{cases}$$

Minimum: $x_i^* = 100, \quad F(x^*) = -300, \quad i = 1(1)n$

Start: $x_i^{(0)} = 0, \quad F(x^{(0)}) = 0, \quad i = 1(1)n$

(Hyper banana) $F_2(x) = \sum_{i=2}^{n} \left[\left(x_1 - x_i^2\right)^2 + (x_i - 1)^2 \right] \quad n = 5$

Minimum: $x_i^* = 1, \quad F(x^*) = 0, \quad i = 1(1)n$

Start: $x_i^{(0)} = 10, \quad F(x^{(0)}) = 32724, \quad i = 1(1)n$

(Sphere) $F_3(x) = \sum_{i=1}^{n} x_i^2, \quad n = 5$

Minimum: $x_i^* = 0, \quad F(x^*) = 0, \quad i = 1(1)n$

Start: $x_i^{(0)} = 10$， $F(x^{(0)}) = 500$， $i = 1(1)n$

(Schwefel) $F_4(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$， $n = 5$

Minimum: $x_i^* = 0$， $F(x^*) = 0$， $i = 1(1)n$

Start: $x_i^{(0)} = 10$， $F(x^{(0)}) = 5500$， $i = 1(1)n$

A typical ES utilizes Guassian mutation as the primary search operator. That means the variance is generated by normal (Gaussian) distribution. In order to compare, we adopt another mutation operator: Cauchy mutation. That mutation operator means the variance is generated by Cauchy distribution (obtained by a variable of normal distribution divided by another variable of normal distribution). Clearly, the search step size of Cauchy mutation is larger than the one of Gaussian mutation. In order to avoid the above case of F_2, we design an improved algorithm called the improved FES(IFES) and the original ESs with Gaussian mutation is called GES. IFES generates two offspring from each parent, one by Cauchy mutation and the other by Gaussian mutation. The better one is then chosen as the offspring. In this study, the number of parent is 10, the number of descendants is 100, the selection operator is (μ, λ) selection, the recombination operator is discrete recombination of pairs of parents. The following table shows the results of the studies, taken over 20 trials for each function respectively.

Table 1. Comparison between IFES and FES on F_1 - F_4. All results have been averaged over 20 runs, where "Mean Best" indicates the mean best function values found in the last generation.

Function	No. of Gen.	IFES Mean Best	FES Mean Best
F_1	50	-295.43	-290.35
F_2	50	2.43239e-06	0.362848
F_3	50	2.73408e-12	8.78729e-12
F_4	50	1.72527e-11	5.13986e-10

From above table, we can see that the improved algorithm (IFES) can gain efficiencies for every function somewhat, especially for F_2.

6 Conclusion

Except genetic operators, there are other strategy parameters to impact on the balance. For example, the population size also has impact on the diversity of the population, so it can impact the balance as well. Also we can change the object function by increasing or decreasing penalty coefficients for violated constraints to strike the tradeoff.

As EAs implement the idea of evolution, and as evolution itself must have evolved to reach its current state of sophistication, it is natural to expect adaptation to be used in not only for finding solutions to a problem, but also for tuning the algorithm to strike the balance for efficiency. In EAs, not only do we need to choose the algorithm, representation and operators for the problem, but we also need to choose parameter values and operator probabilities for the evolutionary algorithm, so that the algorithm will find the solution finally and, what is also important, find it efficiently. Researchers have used various ways of finding good values for the strategy parameters to strike the balance between exploration and exploitation, as these can significantly affect the performance of the algorithm. They tried to modify the values of strategy parameters during the run of the algorithm; it is possible to do this by using some (possibly heuristic) rule (deterministic) [11], by taking feedback from the current state of the search (adaptive), for example, Rechenberg's '1/5 success rule', or by employing some self-adaptive mechanism [12]. Clearly, by changing these values while the algorithm is searching for the solution of the problem, further efficiencies can be gained.

Evolutionary Computation can solve many large search space problems that the traditional methods cannot handle. But if we want to make the Evolutionary Algorithms more efficient, we must handle this problem from the main part of EC: Genetic Search. By analyzing the evolution progress of genetic search, we find two factors that impact the behavior or the performance of EAs: Exploration and Exploitation. Through the analysis of genetic operators and discussion of implementation, we can conclude that in the early stage of the EA, we'd better focus on exploration, because it can avoid the local optimum and premature convergence; in the later stage of the EA, we have to emphasize on exploitation, for its ability to improve the solution quality. Once the balance between the two factors is struck, further efficiencies of EAs can be gained.

Acknowledgements. This work was supported by the National Natural Science Foundation of China (No.60772163) and the Shenzhen Scientific Technology Project (No. SY200806300270A). Thanks especially give to the anonymous reviewers for their valuable comments.

References

1. Baeck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
2. Fogel, D.B.: An Introduction to Simulated Evolutionary Optimization. IEEE Trans. on Neural Networks 5(1), 3–4 (1994)
3. Baeck, T., Schwefel, H.-P.: Evolutionary Computation: An Overview. In: Proc. of the 1996 IEEE Int'l. Conf. on Evolutionary Computation (ICEC 1996), Nagoya, Japan, pp. 20–29. IEEE Press, New York (1996)
4. Yao, X., Liu, Y.: Fast Evolutionary Programming. In: Fogel, L.J., Angeline, P.J., Baeck, T. (eds.) Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming, pp. 257–266. The MIT Press, Cambridge (1996)
5. Yao, X., Lin, G., Liu, Y.: An Analysis of Evolutionary Algorithms Base on Neighborhood and Step Sizes. In: Angeline, P.J., McDonnell, J.R., Reynolds, R.G., Eberhart, R. (eds.) EP 1997. LNCS, vol. 1213, pp. 297–307. Springer, Heidelberg (1997)

6. Kappler, C.: Are Evolutionary Algorithms Improved by Large Mutations? In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 346–355. Springer, Heidelberg (1996)
7. Yao, X., Liu, Y.: Fast Evolutiona Strategies. Control and Cybernetics 26(3), 467–496 (1997)
8. Baeck, T., Gudolph, G., Schwefel, H.-P.: Evolutionary Programming and Evolution Strategies: Similarities and Differences. In: Fogel, D.B., Atmar, W. (eds.) Proc. of the Second Ann. Conf. on Evol. Prog., pp. 11–22. Evolutionary Programming Society, La Jolla
9. Davis, L.: Genetic Algorithms and Simulated Annealing, pp. 1–11. Morgan Kaufmann Publishers, Los Altos (1987)
10. Chellapilla, K.: Combining mutation operators in evolutionary programming. IEEE Trans. on Evolutionary Computation 2(3), 91–96 (1996)
11. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, Heidelberg (1996)
12. Schwefel, H.-P.: Evolution and Optimum Seeking. Sixth Generation Computer Technology Series. Wiley, Chichester (1995)
13. Guo, T.: Evolutionary Computation and Optimization. PhD thesis, Wuhan University, Wuhan (1999)
14. Guo, T., Kang, L.: A New Evolutionary Algorithm for Function Optimization. Wuhan University Journal of Natural Sciences 4(4), 404–419 (1999)
15. Lin, G., Kang, L., Chen, Y., McKay, B., Sarker, R.: A Self-adaptive Mutations with Multiparent Crossover Evolutionary Algorithm for Solving Function Optimization Problems. In: Kang, L., Liu, Y., Zeng, S. (eds.) ISICA 2007. LNCS, vol. 4683, pp. 157–168. Springer, Heidelberg (2007)
16. Lin, G., Kang, L., Chen, Y., McKay, B., Sarker, R.: Comparing the Selective Pressure of Different Selection Operators Progress in Intelligence Computation and Applications, pp. 41–45 (2007)