Clonal and Cauchy-mutation Evolutionary Algorithm for Global Numerical Optimization

Jing Guan[1] and Ming Yang[2]

[1] Institute for Pattern Recognition and Artificial Intelligence, Huazhong University of Science and Technology, Wuhan, 430074, China
g_jing0414@yahoo.com.cn
[2] School of Computer Science, China University of Geosciences, Wuhan, 430074, China
yangming0702@gmail.com

Abstract. Many real-life problems can be formulated as numerical optimization of certain objective functions. However, for an objective function possesses numerous local optima, many evolutionary algorithms (EAs) would be trapped in local solutions. To improve the search efficiency, this paper presents a clone and Cauchy-mutation evolutionary algorithm (CCEA), which employs dynamic clone and Cauchy mutation methods, for numerical optimization. For a suit of 23 benchmark test functions, CCEA is able to locate the near-optimal solutions for almost 23 test functions with relatively small variance. Especially, for f_{14}-f_{23}, CCEA can get better solutions than other algorithms.

Keywords: numerical optimization**,** evolutionary algorithm, clone, Cauchy.

1 Introduction

As optimization problems exist widely in all domains of scientific research and engineering application, research on optimization methods is of great theoretical significance and practical value. Most function optimization problems, such as the optimization of fuzzy system structure and parameters or the optimization of control system, are all multi-peaks function optimization problems，this kind of problems is to search the best point, which is the minimum point commonly from all the search space. For those problems to get the maximum point, they can make minus and are changed to search the minimum point. The minimum function optimization problem can be described as follows:

$$\min f\left(x\right), x \in D \tag{1}$$

where f is an N-dimension function, D is a limited space and $D \subseteq R^N$.

A number of evolutionary algorithms have been used to solve these function optimization problems [1-9]. This success is due to EAs essentially are search algorithms based on the concepts of natural selection and survival of the fittest. They guide the evolution of a set of randomly selected individuals through a number of generations in

Z. Cai et al. (Eds.): ISICA 2009, LNCS 5821, pp. 217–227, 2009.

approaching the global optimum solution. Besides that, the fact that these algorithms do not require previous considerations regarding the problem to be optimized and offers a high degree of parallelism is also true.

Fast evolutionary strategy (FES), applying Cauchy mutation in the evolution strategies to generate each new generation, can find near-optima very fast [4]. Especially, evolutionary programming (EP) has been applied with success to many function optimization problems in recent years [3,5,6], such as classical evolutionary programming (CEP) [5] and fast evolutionary programming (FEP) [6]. For CEP, there are some versions with different mutation operators, namely, (a) Gaussian mutation operator (CEP/GMO), designed for fast convergence on convex function optimization; (b) Cauchy mutation operator (CEP/CMO), aimed for effective escape from the local optima; (c) mean mutation operator (CEP/MMO), which is a linear combination of Gaussian mutation and Cauchy mutation. FEP essentially employs a CMO but incorporates the GMO in an effective way, and it performs much better than CEP for multimodal functions with many local minima while being comparable to CEP in performance for unimodal and multimodal functions with only a few local minima. Evolutionary optimization (EO) [7] uses a mutation operator and a selection scheme to evolve a population. Stochastic genetic algorithm (StGA), employing a novel stochastic coding strategy so that the search space is dynamically divided into regions using a stochastic method and explored region-by-region, and in each region, a number of children are produced through random sampling, and the best child is chosen to represent the region, can get very good results for many function optimization problems [8]. Quantum-inspired evolutionary algorithm, inspired by the multiple universes principle of quantum computing, can also solve some function optimization problems effectively [9]. Some other algorithms such as particle swarm optimization (PSO) [7,10-12] and differential evolution (DE) [13-15] are proposed to solve function optimization problems. PSO is a new evolutionary computing scheme, which explores the insect swarm behavior, and seems to be effective for optimizing a wide range of functions. DE is a practical approach to global function optimization that is easy to understand, simple to implement, reliable, and fast. Basically, DE adds the weighted difference between two population vectors to a third vector and is completely self-organizing.

In this paper, a clone and Cauchy-mutation evolutionary algorithm (CCEA) for function optimization is proposed, using dynamic clone, the number of which is direct proportion to “affinity” between individuals, and Cauchy mutation methods.

2 Clone and Cauchy-mutation Evolutionary Algorithm (CCEA)

Suppose that $\boldsymbol{A}(k)=\{A_1(k+1),A_2(k+1),...,A_n(k+1)\}$ is the population at the k generation and n is the size of population.

The individual $\boldsymbol{A}(k)$ is coded by real number:

$$A_i(k) = (x_{i1}, x_{i2}, ..., x_{im}), i = 1, 2, ..., n \tag{2}$$

where m is the dimension of function and $x_{ij} \in [a_j, b_j]$; a_j, b_j is the boundary of x_j.

CCEA clones an individual according to "affinity" between individuals, operates Cauchy mutation on the population and then generation the offspring population. The course of CCEA is as follow:

$$\boldsymbol{A}(k) \xrightarrow{T_c} \boldsymbol{Y}(k) \xrightarrow{T_g} \boldsymbol{A}(k) \cup \boldsymbol{Y}'(k) \xrightarrow{T_s} \boldsymbol{A}(k+1) \tag{3}$$

where T_c is the dynamic clone operation, T_g is the genetic operation and T_s is the selection operation to generate offspring population, $\boldsymbol{Y}(k)$ is the clone of $\boldsymbol{A}(k)$ got by T_c, $\boldsymbol{Y}'(k)$ is the clone population after T_g. The three operations are described as Fig. 1.

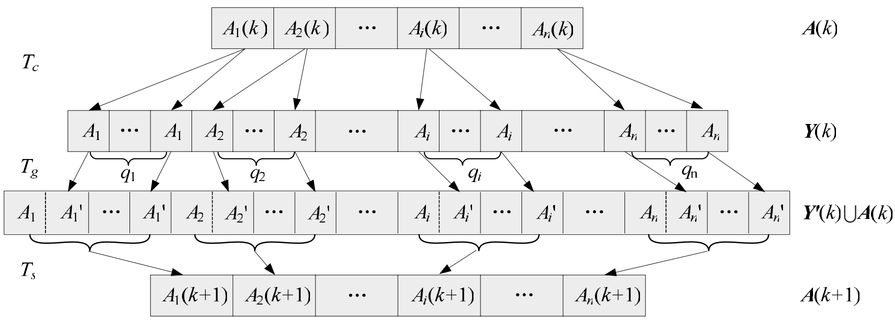

Fig. 1. The three operations of CCEA: T_c , T_g and T_s

2.1 Clone Operation T_c

For an individual, the number of clone got by T_c is determined by the density of its nearby individuals, which is called "affinity". If the density is large, the clone number is small; if the density is small, the number is large. And the number changes with the distribution of population. With this clone operation, domains with smaller density can get larger possibility of genetic operations. So CCEA can search the whole space most possibly and get the global optimal solution.

Suppose that $\boldsymbol{Y}(k)$ is the clone population after T_c. $\boldsymbol{Y}(k)$ is:

$$\boldsymbol{Y}(k) = T_c(\boldsymbol{A}(k)) = \begin{bmatrix} Y_1(k) & Y_2(k) & \cdots & Y_n(k) \end{bmatrix}^{\mathrm{T}} \tag{4}$$

where $Y_i(k) = T_c(A_i(k)) = \boldsymbol{I}_i \times A_i(k) = \begin{bmatrix} A_i(k) \; A_i(k) \; \cdots \; A_i(k) \end{bmatrix}_{1\times q_i}$, $i = 1, 2, ..., n$, $\boldsymbol{I}_i$ is the q_i-dimension row vector in which each element is 1 and the value of q_i is:

$$q_i = \lceil low + (up - low) \times \Theta_i \rceil \tag{5}$$

where *low* and *up* is respectively the minimum and maximum value of clone size, Θ_i is the affinity between A_i and other individuals and $\Theta_i \in [0,1]$:

$$\Theta_i = \frac{d_i - d_{\min}}{d_{\max} - d_{\min}} \tag{6}$$

where d_i is the Euclidian distance between A_i and other individuals A_j:

$$\begin{aligned} d_i &= \min \left\| A_i - A_j \right\|,\ j = 1,2,...,n,\ i \neq j \\ d_{\min} &= \min\{d_i\},\ i = 1,2,...,n \\ d_{\max} &= \max\{d_i\},\ i = 1,2,...,n \end{aligned} \tag{7}$$

From Equation (5) and (6), it can be seen that q_i changes with Θ_i which indicates how dense the population's distribution is near the individual A_i. The more dense distribution, the less Θ_i is, vice versa.

2.2 Cauchy Mutation Operation T_g

This algorithm uses Cauchy mutation, and the one-dimension Cauchy density function is defined by:

$$f(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}\ ,\ (-\infty < x < +\infty) \tag{8}$$

where t>0 is a scale parameter [16], in this paper t=1. The corresponding distribution function is:

$$F(x) = \frac{1}{2} + \frac{1}{\pi} \arctan\left(\frac{x}{t}\right) \tag{9}$$

There are some papers use the Gaussian mutation as their mutation operator, in paper [6], experiments proved that Cauchy density function is similar to Gaussian density function and less than it in vertical direction. The Cauchy distribution changes as slowly as it approaches the horizontal axis, as a result, the variance of the Cauchy distribution is infinite. Fig. 2 shows the difference between Cauchy and Gaussian density functions by plotting them in the same scale.

According to the similarity between Cauchy and Gaussian distribution, especially Cauchy distribution has two sides probability characteristics, the Cauchy distribution can generate a random number far away from origin easily, whose distribution range is wider than that generated from Gaussian mutation, it means Cauchy mutation can jump out of the local optimum quickly.

$\forall A_i(k) \in Y_i(k)$, $A_i(k) = (x_{i1}, x_{i2}, ..., x_{im})$, after the Cauchy mutation operation T_g, the clone individuals of $A_i(k)$ is $\boldsymbol{Y}'(k) = T_g(A_i(k)) = \left[A_i'(k)\ A_i'(k)\ \cdots\ A_i'(k)\right]_{1\times q_i}$, where $A_i'(k) = (x_{i1}', x_{i2}', ..., x_{im}')$. The following is the description of T_g.

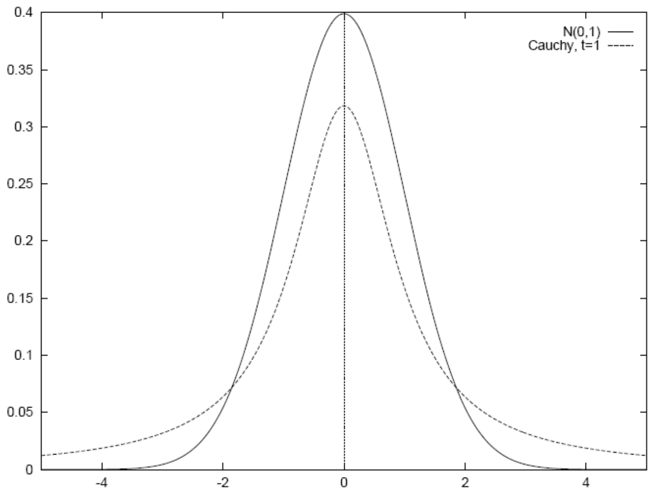

Fig. 2. Comparison between Cauchy and Gaussian density functions

$for\ (1 \le j \le m)$
{
  $if\ (random(0,1)<P_m)$
  {
    $if\ (random(0,1)<0.5)$
      $x'_{ij} = x_{ij} + \delta_{ij}, 1 \le j \le m$;
    $else$
      $x'_{ij} = x_{ij} - \delta_{ij}, 1 \le j \le m$;
    $if\ (x'_{ij} < a_j)\ \ x'_{ij} = a_j$;
    $else\ if\ (x'_{ij} > b_j)\ \ x'_{ij} = b_j$;
  }
}

where $P_m = m^{-(n+1)/(n+t_i)}$, $P_m \in \left[\frac{1}{m}, \sqrt{\frac{1}{m}}\right]$, t is the ranking of $A_i(k+1)$ in the sort sequence according to fitness; δ_{ij} is a Cauchy variable and $\delta_{ij} \in \left[0, r * b_j\right]$.

Relative to the individual of better fitness, the ones of worse fitness can get greater probability. This can guarantee the whole population is good.

2.3 Selection Operation T_s

After Cauchy mutation operation, the offspring population is: $\forall i = 1, 2, ..., n$

$$A_i(k+1) = \begin{cases} B_i(k) & \text{if } B_i(k) \text{ is better than } A_i(k) \\ A_i(k) & \text{else} \end{cases} \tag{10}$$

where $B_i(k) = best\left\{Y'_i(k)\right\} = best\left\{A'_i(k) \mid i = 1, 2, ..., q_i\right\}$.

2.4 The Framework of CCEA

1. Suppose that k is the number of evolution, k=0;
2. Initialize the population $\boldsymbol{A}(k)$ randomly, the size of population is n;
3. Clone operation T_c for each individual in $\boldsymbol{A}(k)$, generate $\boldsymbol{Y}(k)$;
4. Cauchy mutation operation T_g for each individual in $\boldsymbol{Y}(k)$, generate $\boldsymbol{Y}'(k)$;
5. Selection Operation T_s, generate the offspring population $\boldsymbol{A}(k+1)$;
6. $k:=k+1$;
7. If not termination condition, go to step 4;
8. Stop.

3 Experiments

Numerical experiments are conducted to test the effectiveness and efficiency of CCEA. Twenty-three test functions in [6] of three categories are used in experiments, covering a broader range than in some other relevant studies for the purpose to demonstrate the robustness and reliability of the present algorithm.

Table 1 lists the 23 test functions and their key properties. These functions can be divided into three categories of different complexities. f_1-f_7 are unimodal functions, which are relatively easy to optimize, but the difficulty increases as the problem dimension goes high. f_8-f_{13}, representing the most difficult class of problems for many optimization algorithms, are multimodal functions with many local optima, and the number of local minima increases exponentially with the problem dimension [17,18]. f_{14}-f_{23} are likewise multimodal functions, but they only contain a few local optima [18]. The major difference between f_8-f_{13} and f_{14}-f_{23} is that functions f_{14}-f_{23} appear to be simpler than f_8-f_{13} due to their low dimensionalities and a smaller number of local minima. As examples of the three categories, Fig. 3 shows the surface landscapes of f_3, f_8 and f_{23} when the dimension is set to 2. It is interesting to note that some functions possess rather unique features. For instance, f_6 is a discontinuous step function having a single optimum; f_7 is a noisy function involving a uniformly distributed random variable within [0, 1).

Generally speaking, for unimodal functions the convergence rates are of main interest as optimizing such functions to a satisfactory accuracy is not a major issue. For multimodal functions, however, the quality of the final results is more crucial since it reflects the algorithm's ability in escaping from local deceptive optima and locating the desired near-global solution.

In this paper, CCEA is compared with FEP and StGA, where results of FEP and StGA are from [6] and [8]. The parameters of algorithm are set as Table 2. For each test function, 50 runs with different seeds from the random number generator are performed to observe the consistency of the outcome. The results of this comparison are shown in Table 3 and Table 4 with respect to the mean best values found and the standard deviations for each function.

Table 1. The 23 benchmark functions used in experiment, where N is the dimension of function, f_{min} is the minimum value of function, and D is the search space ($\boldsymbol{x} \in \boldsymbol{D}$), the sign "≈" in column f_{min} means the value is approximate

Functions	N	S	f_{min}
f_1	30	$[-100, 100]^N$	0
f_2	30	$[-10, 10]^N$	0
f_3	30	$[-100, 100]^N$	0
f_4	30	$[-100, 100]^N$	0
f_5	30	$[-30, 30]^N$	0
f_6	30	$[-100, 100]^N$	0
f_7	30	$[-1.28, 1.28]^N$	0
f_8	30	$[-500, 500]^N$	-12569.5
f_9	30	$[-5.12, 5.12]^N$	0
f_{10}	30	$[-32, 32]^N$	0
f_{11}	30	$[-600, 600]^N$	0
f_{12}	30	$[-50, 50]^N$	0
f_{13}	30	$[-50, 50]^N$	0
f_{14}	2	$[-65.536, 65.536]^N$	≈1
f_{15}	4	$[-5, 5]^N$	≈0.0003075
f_{16}	2	$[-5, 5]^N$	-1.0316285
f_{17}	2	$[-5, 10] \times [0, 15]$	0.398
f_{18}	2	$[-2, 2]^N$	3
f_{19}	3	$[0, 1]^N$	-3.86
f_{20}	6	$[0, 1]^N$	-3.32
f_{21}	4	$[0, 10]^N$	≈-10.1422
f_{22}	4	$[0, 10]^N$	≈-10.3909
f_{23}	4	$[0, 10]^N$	≈-10.5300

From Table 3, it can be seen that CCEA can get the near-global solution except f_8 and f_9. In the experiment, we found because there are so many local optima, the steps between optima are large, and the probability of generating large Cauchy variables is very small, CCEA can not jump out of local optima and get the global solution. From Table 3 and Table 4, for from f_1 to f_{13}, we can see CCEA have the same performance to FEP and StGA, except f_8 and f_9. For from f_{14} to f_{23}, CCEA performs better than FEP and StGA.

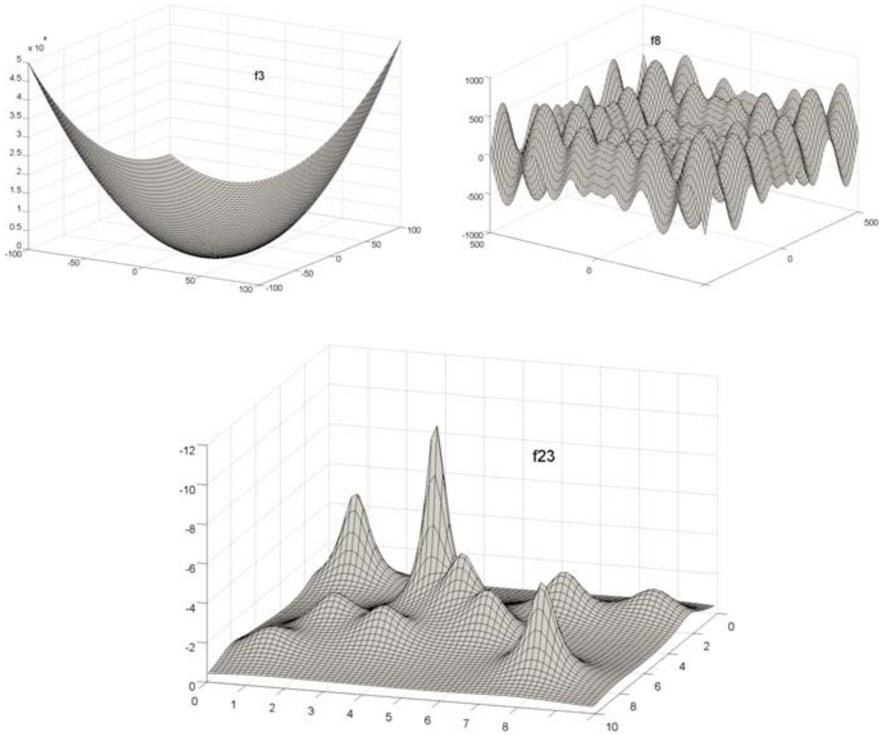

Fig. 3. Graphs of f_3, f_8 and f_{23} with a dimension of 2

Table 2. Algorithm's parameters setting

n	low	up	r_1	t
50	10	20	0.1	1

Table 3. Comparison between CCEA and FEP. All results have been averaged over 50 runs, where "Mean Best"indicates the mean best function values found in the last generation, and "Std Dev" stand for the standard deviation.

TF	Number of generation		Mean Best		Std Dev	
	CCEA	FEP	CCEA	FEP	CCEA	FEP
f_1	10000	1500	5.27×10^{-5}	5.7×10^{-4}	8.99×10^{-6}	1.3×10^{-4}
f_2	50000	2000	2.89×10^{-3}	8.1×10^{-3}	2.49×10^{-4}	7.7×10^{-4}
f_3	50000	5000	4.40×10^{-2}	1.6×10^{-2}	9.46×10^{-3}	1.4×10^{-2}

Table 3. (*Continued*)

TF	Number of generation		Mean Best		Std Dev	
	CCEA	FEP	CCEA	FEP	CCEA	FEP
f_4	50000	5000	6.29×10^{-3}	0.3	3.86×10^{-4}	0.5
f_5	50000	20000	0.2	5.06	0.24	5.87
f_6	500	1500	0	0	0	0
f_7	10000	3000	1.29×10^{-3}	7.6×10^{-3}	1.63×10^{-3}	2.6×10^{-3}
f_8	10000	9000	-8315.98	-12554.5	383.29	52.6
f_9	10000	5000	161.681	4.6×10^{-2}	19.92	1.2×10^{-2}
f_{10}	50000	1500	8.82×10^{-4}	1.8×10^{-2}	1.19×10^{-4}	2.1×10^{-3}
f_{11}	10000	2000	3.89×10^{-6}	1.6×10^{-2}	1.58×10^{-6}	2.2×10^{-2}
f_{12}	10000	1500	4.38×10^{-7}	9.2×10^{-6}	1.36×10^{-7}	3.6×10^{-6}
f_{13}	10000	1500	5.69×10^{-6}	1.6×10^{-4}	1.04×10^{-6}	7.3×10^{-5}
f_{14}	500	100	0.998004	1.22	0	0.56
f_{15}	50000	4000	3.11×10^{-4}	5.0×10^{-4}	3.14×10^{-6}	3.2×10^{-4}
f_{16}	100	100	-1.03163	-1.03	1.49×10^{-6}	4.9×10^{-7}
f_{17}	100	100	0.397889	0.398	1.45×10^{-6}	1.5×10^{-7}
f_{18}	500	100	3	3.02	2.01×10^{-6}	0.11
f_{19}	100	200	-3.86278	-3.86	1.22×10^{-6}	1.4×10^{-5}
f_{20}	100	100	-3.32188	-3.27	4.63×10^{-5}	5.9×10^{-2}
f_{21}	1000	100	-10.1532	-5.52	2.19×10^{-5}	1.59
f_{22}	1000	100	-10.4029	-5.52	1.79×10^{-5}	2.12
f_{23}	1000	100	-10.5364	-6.57	4.10×10^{-5}	3.14

4 Conclusions

This paper introduced a clone and Cauchy-mutation evolutionary algorithm (CCEA), which employs dynamic clone and Cauchy mutation methods, for numerical optimization. With clone operation, domains with smaller density can get larger possibility of genetic operations. So CCEA can search the whole space most possibly, jump out of the local optima, and get the global optimal solution. For a suit of 23 benchmark test functions, CCEA is able to locate the near-optimal solutions for almost 23 test functions with relatively small variance, indicating that the algorithm is both effective and statistically stable.

The future work is to improve the mutation random variables, and to jump out of local optima and locate the global optima with larger possibility.

Table 4. Comparison between CCEA and StGA. All results have been averaged over 50 runs, where "Mean Best"indicates the mean best function values found in the last generation, and "Std Dev" stand for the standard deviation.

TF	Number of generation		Mean Best		Std Dev	
	CCEA	StGA	CCEA	StGA	CCEA	StGA
f_1	10000	30000	5.27×10^{-5}	2.45×10^{-15}	8.99×10^{-6}	5.25×10^{-16}
f_2	50000	17600	2.89×10^{-3}	2.03×10^{-7}	2.49×10^{-4}	2.95×10^{-8}
f_3	50000	23000	4.40×10^{-2}	9.98×10^{-29}	9.46×10^{-3}	6.9×10^{-29}
f_4	50000	32000	6.29×10^{-3}	2.01×10^{-8}	3.86×10^{-4}	3.42×10^{-9}
f_5	50000	45000	0.2	0.04435	0.24	0
f_6	500	1500	0	0	0	0
f_7	10000	25500	1.29×10^{-3}	8.4×10^{-4}	1.63×10^{-3}	1.0×10^{-3}
f_8	10000	1500	-8315.98	-12569.5	383.29	0
f_9	10000	28500	161.681	4.42×10^{-13}	19.92	1.14×10^{-13}
f_{10}	50000	10000	8.82×10^{-4}	3.52×10^{-8}	1.19×10^{-4}	3.51×10^{-8}
f_{11}	10000	52500	3.89×10^{-6}	2.44×10^{-17}	1.58×10^{-6}	4.54×10^{-17}
f_{12}	10000	8000	4.38×10^{-7}	8.03×10^{-7}	1.36×10^{-7}	1.96×10^{-14}
f_{13}	10000	16000	5.69×10^{-6}	1.13×10^{-5}	1.04×10^{-6}	4.62×10^{-13}
f_{14}	500	800	0.998004	1	0	0
f_{15}	50000	30000	3.11×10^{-4}	3.1798×10^{-4}	3.14×10^{-6}	4.7262×10^{-6}
f_{16}	100	4000	-1.03163	-1.03034	1.49×10^{-6}	1.0×10^{-3}
f_{17}	100	5000	0.397889	0.3986	1.45×10^{-6}	6.0×10^{-4}
f_{18}	500	/	3	/	2.01×10^{-6}	/
f_{19}	100	/	-3.86278	/	1.22×10^{-6}	/
f_{20}	100	/	-3.32188	/	4.63×10^{-5}	/
f_{21}	1000	10000	-10.1532	-9.828	2.19×10^{-5}	0.287
f_{22}	1000	4800	-10.4029	-10.40	1.79×10^{-5}	0
f_{23}	1000	8500	-10.5364	-10.450	4.10×10^{-5}	0.037

References

1. Bäck, T., Fogel, D.B., Michalewicz, Z. (eds.): Handbook of Evolutionary Computation. Institute to Physics Publishing (1997)
2. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs, 2nd edn. Springer, New York (1994)
3. Gehlhaar, D.K., Fogel, D.B.: Tuning evolutionary programming for conformationally flexible molecular docking. In: Fogel, L.J., Angeline, P.J., Bäck, T. (eds.) Evolutionary Programming V: Proc. Of the Fifth Annual Conference on Evolutionary Programming, pp. 419–429. MIT Press, Cambridge (1996)

4. Yao, X., Liu, Y., Lin, G.M.: Fast evolutionary strategy. In: Angeline, P.J., Reynods, R., McDonnell, J., Eberhart, R. (eds.) Proc. Evolutionary Programming VI, pp. 151–161 (1997)
5. Chellapilla, K.: Combining mutation operators in evolutionary programming. IEEE Trans. Evol. Comput. 2, 91–98 (1998)
6. Yao, X., Liu, Y., Lin, G.M.: Evolutionary programming made faster. IEEE Trans. on Evolutionary Computation 3, 82–102 (1999)
7. Angeline, P.J.: Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In: Porto, V.W., Saravanan, N., Waagen, D., Eiben, A.E. (eds.) Proc. Evolutionary Programming VII, pp. 601–610 (1998)
8. Tu, Z., Lu, Y.: A robust stochastic genetic algorithm (stga) for global numerical optimization. IEEE Trans. on Evolutionary Computation 8(5), 456–470 (2004)
9. André, V., da Curz, A., Vellasco, M.M.B.R.: Quantum-Inspired Evolutionary Algorithm for Numerical Optimization. In: 2006 IEEE Congress on Evolutionary Computation, pp. 9181–9187 (2006)
10. Clerc, M., Kennedy, J.: The particle swarm-explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. on Evolutionary Computation 6(1), 58–73 (2002)
11. Parsopoulos, K.E., Vrahatis, M.N.: Recent approaches to global optimization problems through particle swarm optimization. Natural Computing, 235–306 (2002)
12. Andrews, P.S.: An Investigation into Mutation Operators for Particle Swarm Optimization. In: 2006 IEEE Congress on Evolutionary Computation, pp. 3789–3796 (2006)
13. Storn, R., Price, K.V.: Differential evolution-A simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization 11, 341–359 (1997)
14. Lampinen, J., Zelinka, I.: Mixed Variable Non-linear Optimization by Differential Evolution. In: Proceedings of Nostradamus 1999 2nd International Prediction Conference, pp. 45–55 (1999)
15. Price, K., Storm, R., Lampinen, J.: Differential Evolution-A Practical Approach to Global Optimization. Springer, Heidelberg (2005)
16. Feller, W.: An Introduction to Probability Theory and Its Applications, 2nd edn., vol. 2, p. 51. John Wiley & Sons, Inc., Chichester (1971)
17. Törn, A., Žilinskas, A.: Global Optimization. LNCS, vol. 350. Springer, Heidelberg (1989)
18. Schwefel, H.P.: Evolution and Optimum Seeking. Wiley, New York (1995)