Differential Evolution using a Localized Cauchy Mutation Operator

Radha Thangraj[1], Millie Pant[1], Ajith Abraham[2], Kusum Deep[1], Vaclav Snasel[3]
[1] Indian Institute of Technology Roorkee, India
[2] Machine Intelligence Research Labs (MIR Labs),
Scientific Network for Innovation and Research Excellence, USA
[3]VSB Technical University of Ostrava, Czech Republic
t.radha@ieee.org, millifpt@iitr.ernet.in, ajith.abraham@ieee.org, kusumfma@iitr.ernet.in, vaclav.snasel@vsb.cz

***Abstract*— In the present work, we propose a new variant of basic DE algorithm called CMDE-G which uses Cauchy mutation (CM) operator. In this algorithm, at the end of every generation, CM is applied as a local search mechanism to explore the neighborhood of the best individual in the population. The performance of CMDE-G algorithm is analyzed on a set of 10 standard benchmark problems and four nontraditional composite functions. Simulation results show that the proposed algorithm helps in improving the solution quality besides maintaining a good convergence rate.**

***Keywords*—Differential Evolution, mutation, cauchy distribution, local search.**

I. INTRODUCTION

Differential Evolution (DE), an optimization technique, is an exceptionally simple and easy to use evolutionary strategy. It is significantly faster and robust at numerical optimization and is more likely to find a function's true global optimum [1]. Despite having several attractive features, practical experiences shows that DE sometimes does not perform up to the expectations. Like most of the population based search techniques the driving force behind the success of DE is the balance between the exploration (diversification) and exploitation (intensification) processes. If these two are not well defined than problems like premature convergence or stagnation of population may take place.

Several modifications have been made in the structure of basic DE to improve its performance. Some interesting modifications include parameter adaption strategy for DE by Zaharie [2], Abbas [3] proposed a self adaptive crossover rate for multiobjective optimization problems, Omran et al. [4] introduced a self adaptive scaling factor parameter *F*, Brest et al. [5] proposed SADE, which encoded control parameters F and Cr into the individuals and evolved their values by using two new probabilities. Teo [6] proposed an attempt at self-adapting the population size parameter in addition to self-adapting crossover and mutation rates. Yang et al. [7] proposed a self adaptive differential evolution algorithm with neighborhood search (SaNSDE). SaNSDE proposes three self-adaptive strategies: self adaptive choice of the mutation strategy between two alternatives, self-adaptation of the scale factor *F*, and self-adaptation of the crossover rate *Cr*. Qin et al. [8] proposed a Self-adaptive DE algorithm (SaDE), where the choice of learning strategy and the two control parameters F and CR are not required to be pre-defined.

In [9] and [10], Pant et al. suggested new mutation strategies based on Laplace probability distribution and Quadratic Interpolation respectively. Several developments in DE algorithm design and application can be found in [11]. In continuation to the techniques of improving the performance of DE, in the present study we propose a modified version of DE called CMDE-G. In CMDE-G Cauchy mutation is not added just to induce a small perturbation in the population but is included to provide a local search effect.

The remaining of the paper is organized as follows: Sections II and III explain the original DE and the proposed CMDE-G algorithms respectively. In section IV, experimental settings and benchmark problems are given; the numerical results are analyzed in section V. Finally the paper concludes with Section VI.

II. DIFFERENTIAL EVOLUTION

In DE, a population of potential solutions within an n-dimensional search space, a fixed number of vectors, are randomly initialized and then evolved over time to explore the search space and locate the minima of the objective function. At each iteration, called generation, new vectors are generated by the combination of vectors randomly chosen from the current population (mutation). The out coming vectors are then mixed with a predetermined target vector to produce a trial vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the objective function. This last operator is referred to as a selection.

DE shares a common terminology of selection, crossover and mutation operators with GA however it is the application of these operators that make DE different from GA; while, in GA crossover plays a significant role, it is the mutation operator which affects the working of DE [12].

The working of basic DE may be described as follows:

For a D-dimensional search space, each target vector $x_{i,g}$, a mutant vector is generated by

$$V_{i,g} = X_{r_1,g} + F*(X_{r_2,g} - X_{r_3,g}) \quad (1)$$

978-1-4244-6588-0/10/$25.00 ©2010 IEEE

where $r_1, r_2, r_3 \in \{1,2,....,NP\}$ are randomly chosen integers, must be different from each other and also different from the running index i. F (>0) is a scaling factor which controls the amplification of the differential evolution $(x_{r_2,g} - x_{r_3,g})$. In order to increase the diversity of the perturbed parameter vectors, crossover is introduced [13]. The parent vector is mixed with the mutated vector to produce a trial vector $u_{ji,g+1}$,

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1} & if \quad rand_j \le Cr \vee j = k \\ x_{j,i,G} & otherwise \end{cases} \qquad (2)$$

where j = 1, 2,……, D; $rand_j \in [0,1]$; CR is the crossover constant takes values in the range [0, 1] and $j_{rand} \in (1,2,.....,D)$ is the randomly chosen index.

Selection is the step to choose the vector between the target vector and the trial vector with the aim of creating an individual for the next generation.

III. Cauchy Mutated Differential Evolution

The use of mutation operation is not new to the field of evolutionary algorithms. Its main aim is to introduce a small perturbation in the population from time to time so as to maintain its diversity. Most of the times the mutation operation is applied according to some fixed probabilistic rule. Also the number of times mutation will take place is also predefined. In the past few years mutation operations based on different probability distributions (like Normal, Gaussian, and Cauchy etc) have become quite popular. The present study attempts to use the Cauchy Mutation operator as a local search strategy.

Cauchy Mutated Differential Evolution (CMDE-G) starts like the basic DE algorithm using the same mutation equation as given in the previous section to generate the perturbed mutant vector. The process of generating the trial vector and selecting the fitter candidate for the next generation are also same as that of basic DE algorithm. Once the selection process is complete i.e. at the end of every iteration we search the neighborhood of the best (or global best) particle say X_{best} with the CM operator. If after mutation the solution quality is improved, then it is applied again to see if the solution can be improved any further. This process continues till we keep getting better solution. In case there is no improvement in the solution, then the algorithm moves to the next iteration.

At the end of every iteration, mutation is defined as:

$$X'_{best} = X_{best} + C^* | X_{r_1} - X_{r_2} | \qquad (3)$$

Where X_{best} is the global best particle, **C** is the Cauchy distributed random number and $r_1, r_2 \in \{1,2,...., NP\}$ are randomly chosen integers, different from each other and also different from the global best particle. The flowchart of CMDE-G algorithm is given in Fig. 1.

IV. Experimental Settings and Benchmark Problems

In order to make a fair comparison of DE and CMDE-G algorithms, we fixed the same seed for random number generation so that the initial population is same for both the algorithms. The population size is taken as 100 for all the test problems. The crossover rate and scaling factor F are fixed at 0.2 and 0.5 respectively. For each algorithm, the maximum number of iterations allowed was set to 5000 and the error goal was set as 1*e-04. A total of 30 runs for each experimental setting were conducted and the average fitness along with the average number of function evaluations (NFE), time taken and number of generations (GNE) of the best solutions throughout the run were recorded. The algorithms were programmed using Developer C++ and were executed on a Pentium IV PC.

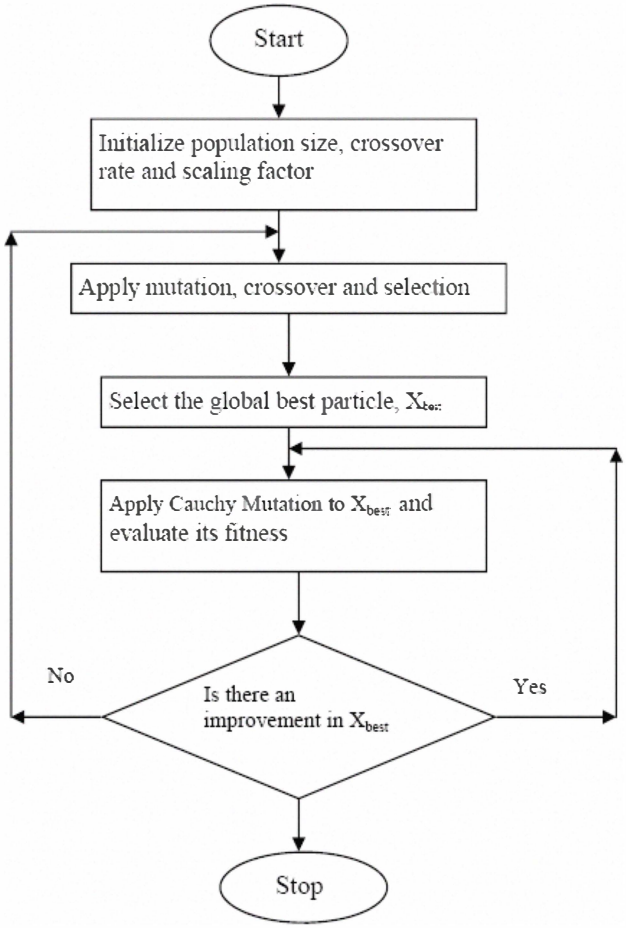

Figure 1. Flowchart of CMDE-G algorithm

In order to check the compatibility of the proposed CMDE-G algorithm we have tested it on a suite of 10 benchmark problems (given in Table I) and 4 nontraditional composite functions [14]. The test bed comprises of a variety of problems ranging from a simple spherical function to highly multimodal functions with several local and global optima. The four composite functions F_4, F_1, F_5 and F_6, taken from CEC 2008 benchmarking problems [14], are here marked as CF_1, CF_2, CF_3 and CF_4 respectively. All these functions are scalable, shifted and multi-modal functions containing a large number of local optima except CF_2. CF_2 is a unimodal function. The

global optimum of both CF_1 is f(x*) = -330, CF_2 is f(x*) = -450, CF_3 is f(x*) = -180 and that for CF_4 is f(x*) = -140. For more details on composite functions and other benchmark problems of similar type the interested reader may please refer to [14]. All the test problems are tested for two different dimensions 25 and 50.

V. Numerical Results and Discussion

A. Analysis of traditional benchmark problems

1) Performance Analysis I: The proposed CMDE-G is compared with basic DE on traditional benchmark problems by using the standard performance metrics like average fitness function value, standard deviation, average number of function evaluations (NFE) and CPU time (in sec) etc. the corresponding results for dimensions 25 and 50 are given in Tables II and III respectively.

From Table II, we can see that CMDE-G is either superior or at par with basic DE for all the test cases in comparison of all the performance measures. If we compare the performance measures one by one then from the comparison of average fitness function value it can be seen that CMDE-G gave better result than DE in 8 test cases out of 10 cases tried. For the remaining two test cases, f_4 and f_6, both algorithms gave same results. The total NFE taken by basic DE for solving the 10 benchmark problems is 1138640 while with CMDE-G it is 957926.2 showing that there is an improvement is around 16%. Likewise an improvement in average CPU time is around 19%. However the success rate for both the algorithms is 90%.

From Table III, where the results are recorded for dimension 50, once again we can see the better performance of the proposed CMDE-G algorithms. Here we can see that CMDE-G outperformed the basic DE in 9 out of 10 test problems in terms of average fitness function value. The total number of function evaluations comes out to be 1521134 for CMDE-G in comparison to 1781363 as obtained by DE, showing an improvement of around 15%. The total time taken by CMDE-G is 173.4whereas the total time taken by DE is 198.32 (an improvement of 13%). Also, in terms of total numbers of generations, the performance of CMDE-G is better than basic DE for problems of dimension 25 and 50. Convergence curves of selected benchmark problems are illustrated in Figures 2 and 3.

2) Performance Analysis II: Performance Index: To further compare the consolidated performance of CMDE-G with original DE with respect to success rate and average number of function evaluations of successful runs both, the value of performance index (PI) [15] is computed. This index gives a weighted importance to the success rate and the average number of function evaluations of successful runs.

The value of this performance index for a computational algorithm under comparison is given by

$$PI = \frac{1}{N}\sum_{i=1}^{N_p}\left(k_1\alpha_1^i + k_2\alpha_2^i + k_3\alpha_3^i\right)$$

where $\alpha_1^i = \frac{Sr^i}{Tr^i}$;

$$\alpha_2^i = \begin{cases} \frac{Mf^i}{Af^i}, & if\ Sr^i > 0 \\ 0 & if\ Sr^i = 0 \end{cases}$$

$$\alpha_3^i = \begin{cases} \frac{Mt^i}{At^i}, & if\ Sr^i > 0 \quad i = 1,2,....,N_p \\ 0 & if\ Sr^i = 0 \end{cases}$$

Sr^i = Number of successful runs of i^{th} problem

Tr^i = Total number of runs of i^{th} problem

Mf^i = Minimum of average number of function evaluations of successful runs used by all algorithms in obtaining the solution of i^{th} problem

Af^i = Average number of function evaluations of successful runs used by an algorithm in obtaining the solution of i^{th} problem

Mt^i = Minimum of average time used by all algorithms in obtaining the solution of i^{th} problem

At^i = Average computational time used by an algorithm in obtaining the solution of i^{th} problem

N_p = Total number of problems analyzed.

k_1 ,k_2*and* k_3 ($k_1 + k_2 + k_3 = 1$ and $0 \le k_1, k_2, k_3 \le 1$) are the weights assigned to success rate and average number of function evaluations of successful runs, respectively. From above definition it is clear that PI is a function of k_1, k_2 and k_3. Since, $k_1 + k_2 + k_3 = 1$, one of k_i, $i = 1, 2, 3$ could be eliminated to reduce the number of dependent variables from the expression of PI. But it is still difficult to analyze the behavior of PI, because the surface plots of PI for DE and CMDE-G are overlapping and it is difficult to visualize them. So, we adopt the same methodology as given in [15] i.e., equal weights are assigned to two terms at a time in the PI expression. This way PI becomes a function of one variable. The resultant cases are as follows

$$k_1 = W, k_2 = k_3 = \frac{1-W}{2}, 0 \le W \le 1$$

$$k_2 = W, k_1 = k_3 = \frac{1-W}{2}, 0 \le W \le 1$$

$$k_3 = W, k_1 = k_2 = \frac{1-W}{2}, 0 \le W \le 1$$

The PI is obtained for DE and CMDE-G for all the benchmark problems and is shown in Figure 4 -6. It is clear that the proposed CMDE-G outperforms the DE.

TABLE I NUMERICAL BENCHMARK PROBLEMS

Function	Function Definition	Range	Min.Value
Rastringin Function	$f_1(x)=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10)$	[-5.12,5.12]	0
Sphere Function	$f_2(x)=\sum_{i=1}^{n}x_i^2$	[-5.12,5.12]	0
Griewank Function	$f_3(x)=\frac{1}{4000}\sum_{i=0}^{n-1}x_i^2+\sum_{i=0}^{n-1}\cos(\frac{x_i}{\sqrt{i+1}})+1$	[-600,600]	0
Step function	$f_4(x)=\sum_{i=0}^{n-1}\lfloor x_i+1/2\rfloor^2$	[-100, 100]	0
Noisy Function	$f_5(x)=(\sum_{i=0}^{n-1}(i+1)x_i^4)+rand[0,1]$	[-1.28,1.28]	0
Schwefel Function	$f_6(x)=-\sum_{i=1}^{n}x_i\sin(\sqrt{\lvert x_i\rvert})$	[-500,500]	-418.9829*n
Ackley Function	$f_7(x)=20+e-20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))$	[-32,32]	0
Generalized penalized function 1	$f_8(x)=\frac{\pi}{n}\{10\sin^2(\pi y_1)+\sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(y_{i+1}\pi)]+(y_n-1)^2\}+\sum_{i=1}^{n}u(x_i,10,100,4)$	[-50, 50]	0
Generalized penalized function 2	$f_9(x)=(0.1)\{\sin^2(3\pi x_1)+\sum_{i=1}^{n-1}((x_i-1)^2(1+\sin^2(3\pi x_{i+1})))+(x_n-1)^2(1+\sin^2(2\pi x_n))\}+\sum_{i=0}^{n-1}u(x_i,10,100,4)$	[-50, 50]	0
Axis parallel hyper ellipsoid	$f_{10}(x)=\sum_{i=1}^{n}ix_i^2$	[-5.12,5.12]	0

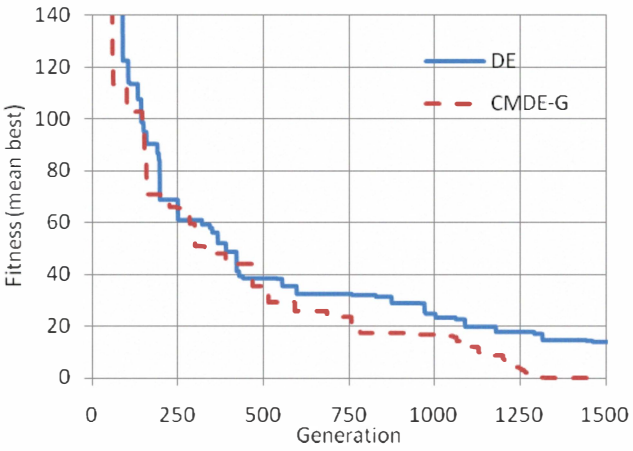

Figure 2. Convergence curves of DE and CMDE-G algorithms for function f_1

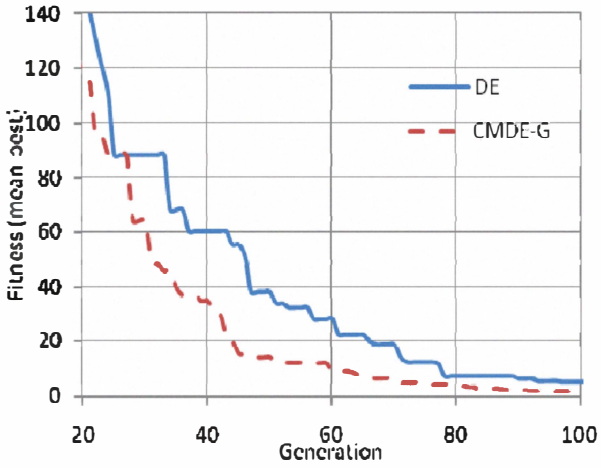

Figure 3. Convergence curves of DE and CMDE-G algorithms for function f_3

TABLE II COMPARISON RESULTS OF DE AND CMDE-G FOR DIMENSION 25

Function	DE			CMDE-G		
	Mean	Std	GNE	Mean	Std	GNE
f_1	4.7217e-6	7.7152e-7	2143.6	3.1604e-6	7.7406e-7	1547
f_2	3.3099e-6	6.2714e-7	397.4	3.0763e-6	1.1412e-6	278.4
f_3	3.6598e-6	6.7250e-7	604.2	3.0401e-6	9.0940e-7	408.8
f_4	0	0	307.8	0	0	208.4
f_5	0.0026	0.0005	5001	0.0018	0.0003	5001
f_6	-10474.6	5.0359e-7	774.4	-10474.6	9.3078e-7	545.2
f_7	1.0546e-5	1.5410e-6	714.6	1.0516e-5	1.2826e-6	497.2
f_8	4.8087e-6	6.8572e-7	486	3.1125e-6	6.1420e-7	326
f_9	3.6033e-6	5.4496e-7	501.4	3.5533e-6	6.0289e-7	347.4
f_{10}	3.5686e-6	2.9750e-7	447	3.3257e-6	2.7893e-7	314.6
Function	DE			CMDE-G		
	NFE	Time(sec)	SR	NFE	Time(sec)	SR
f_1	214460	13.2	100	156364	9.4	100
f_2	39840	2.2	100	28234	1.6	100
f_3	60520	3.8	100	41409.6	2.4	100
f_4	30880	0.4	100	21153.8	0.2	100
f_5	500100	27.2	0	505104	26.4	0
f_6	77540	0.8	100	55187.8	0.6	100
f_7	71560	4.4	100	50341	3.2	100
f_8	48700	5.2	100	33038.4	3.4	100
f_9	50240	6.0	100	35203	4.4	100
f_{10}	44800	2.6	100	31890.6	2	100

TABLE III COMPARISON RESULTS OF DE AND CMDE-G FOR DIMENSION 50

Function	DE			CMDE-G		
	Mean	Std	GNE	Mean	Std	GNE
f_1	102.747	6.3651	5000	74.0724	6.9485	5001
f_2	8.24e-5	1.21e-5	675	7.4065e-6	1.7241e-6	464.2
f_3	8.40e-5	1.56e-5	977	7.6044e-6	1.5238e-6	631.6
f_4	0.00000	0.00000	603	0	0	338.8
f_5	0.01273	0.00234	5000	0.0054	0.0007	5001
f_6	-20669.3	171.058	3427	-20949.1	1.5313e-6	1116
f_7	0.00021	2.46e-5	1141	2.033e-5	1.2523e-6	791.2
f_8	8.62e-5	1.41e-5	931	8.2979e-6	9.5470e-7	556.6
f_9	8.11e-5	1.18e-5	951	7.8884e-6	1.3465e-6	605.4
f_{10}	8.19e-5	1.06e-5	810	7.6930e-6	1.2614e-6	545
Function	DE			CMDE-G		
	NFE	Time (sec)	SR	NFE	Time(sec)	SR
f_1	500100	55.33	0	505127	60.2	0
f_2	67696	7.1	100	47004.4	5.2	100
f_3	97823	12.3	100	63918.4	8.0	100
f_4	60406	1.4	100	34326.4	0.6	100
f_5	500100	52.03	0	505105	53	0
f_6	171432	5.33	100	112842	2.4	100
f_7	114207	13.16	100	80043	9.8	100
f_8	93206	20.67	100	56332.6	12.8	100
f_9	95213	22.6	100	61265.6	15	100
f_{10}	81180	8.4	100	55169.6	6.4	100

TABLE IV RESULTS OF COMPOSITE FUNCTIONS FOR DIMENSIONS 25 AND 50

Dimension 25				
Function	DE		CMDE-G	
	NFE	Time (sec)	NFE	Time (sec)
CF_1	185940	10.4	131622	7.6
CF_2	54160	0.4	38804.6	0.1
CF_3	59380	3.8	44041.8	2.8
CF_4	72140	4.2	51556.2	3.2
Dimension 50				
Function	DE		CMDE-G	
	NFE	Time (sec)	NFE	Time (sec)
CF_1	500100	55.2	505128	57
CF_2	107360	1.0	64058	0.6
CF_3	108020	13.2	65753.2	8.0
CF_4	136880	16.6	81498	9.8

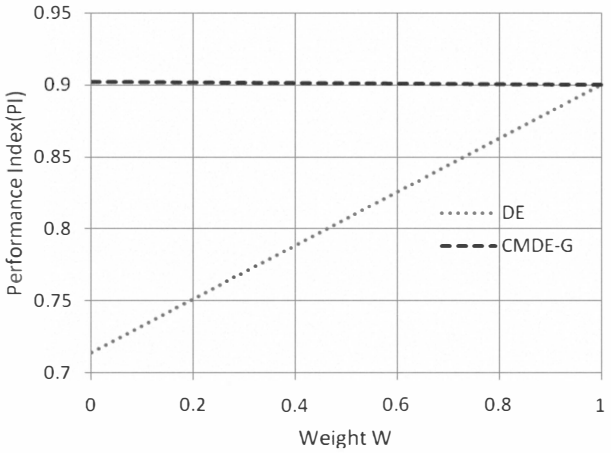

Figure 4. Performance Index when k_1=W, $k_2 = k_3 = (1-W)/2$

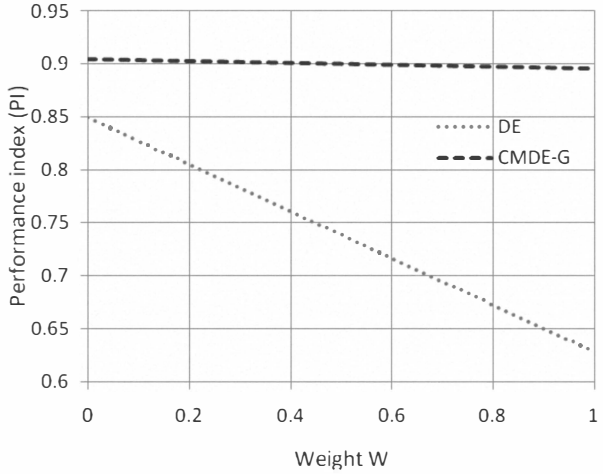

Figure 5. Performance Index when k_2=W, $k_1 = k_3 = (1-W)/2$

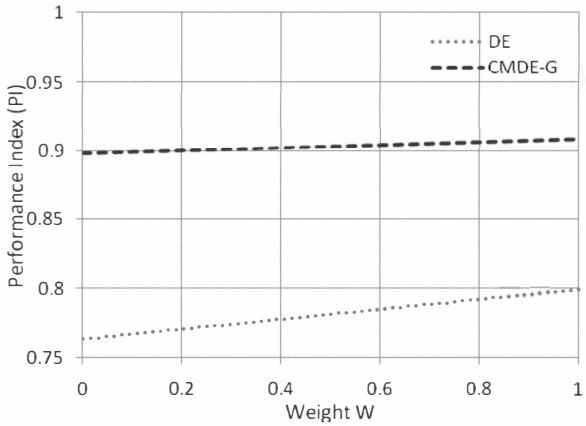

Figure 6. Performance Index when k_3=W, $k_1 = k_2 = (1-W)/2$

B. *Results of composite functions*

Composite Functions were specially designed to test the efficiency of a global optimization algorithm. The numerical results for these functions ($CF_1 - CF_4$) are given in Table IV. Here From this table also, we can see that the proposed CMDE-G algorithm is superior with basic DE for all the dimensions. For dimension 25, the total number of function evaluations for solving the four problems comes out to be 266024 for CMDE-G in comparison to 371620 as obtained by DE, which implies that there is an improvement of 28%. Similarly, the total time taken by CMDE-G is 13.7 whereas the total time taken by DE is 18.8 (an improvement of 27%). Similarly for dimension 50, CMDE-G gave a noticeable percentage of improvement of about 15% in terms of NFE and an improvement of around 41% in terms of average CPU time in comparison to basic DE algorithm.

VI. CONCLUSION

In the present study we proposed a modified version of DE called CMDE-G where Cauchy mutation operator is applied. In most of evolutionary algorithms mutation operator is applied according to some fixed probabilistic rule. In CMDE-G We do not have to fix a mutation probability in the beginning of the algorithm and secondly we do not have to specify the number of times mutation is to be applied in a particular generation. It is a sort of intelligent DE which applies the mutation as per the requirement and not according to some probabilistic rule. If there is an improvement in the fitness function value than the mutation is repeated otherwise the algorithm enters the next generation. The proposed algorithm is validated on a set of ten standard benchmark problems and four composite functions taken from the test suite of CEC2008 benchmark problems. Its comparison with classical DE shows that the use of Cauchy mutation in the form of a local strategy may help in improving the performance of basic DE. In future we intend the compare the performance of CMDE-G with other sophisticated and recent versions of DE. Also we plan to test its efficiency on real life and constrained problems.

REFERENCES

[1] G Price, K., and Storn, R. "Differential evolution—A simple evolution strategy for fast optimization", *Dr. Dobb's Journal*, Vol. 22, pp. 18–24, 1997.

[2] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," In D.Matousek, P. Osmera (eds.), *Proc. of MENDEL 2003, 9th International Conference on Soft Computing,* Brno, Czech Republic, pp. 41-46, June 2003.

[3] H. Abbass, "The self-adaptive pareto differential evolution algorithm," in *Proc. of the 2002 Congress onEvolutionary Computation*, pp. 831-836, 2002.

[4] M. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution, computational intelligence and security," *PT I, Proceedings Lecture Notes In Artificial Intelligence* 3801, pp. 192-199, 2005.

[5] J. Brest, S. Greiner, B. Boškovic, M. Mernik, and V. Žumer, "Self-adapting Control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, Vol. 10 (6), pp. 646 – 657, 2006.

[6] J. Teo, " Exploring Dynamic Self-adaptive Populations in Differential Evolution", Soft Computing - A Fusion of Foundations, Methodologies and Applications, Vol. 10 (8), pp. 673 – 686, 2006.

[7] Yang, Z., Tang, K. and Yao, X. , "Self-adaptive Differential Evolution with Neighborhood Search", In Proc. IEEE Congress on Evolutionary Computation, Hong Kong, pp. 1110-1116, 2008.

[8] Qin, A. K., Huang, V. L. and Suganthan, P. N., "Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization", IEEE Transactions on Evolutionary Computations, Vol. 13 (2), pp. 398 – 417, 2009.

[9] Pant, M., Thangaraj, R., Abraham, A. and Grosan, C., "Differential Evolution with Laplace Mutation Operator, IEEE Congress on Evolutionary Computation, Norway, pp. 2841-2849, 2009.

[10] M. Pant, R. Thangaraj and V. P. Singh, "A New Differential Evolution Algorithm for Solving Global Optimization Problems", Int. Conf. on Advanced Computer Control, Singapore, IEEE Computer Society Press, pp. 388 – 392, 2009.

[11] U. K. Chakraborty (Ed.) *Advances in Differential Evolution*, Springer-Verlag, Heidelberg, 2008.

[12] D. Karaboga and S. Okdem, "A simple and Global Optimization Algorithm for Engineering Problems: Differential Evolution Algorithm", Turk J. Elec. Engin., Vol. 12(1), pp. 53 – 60, 2004.

[13] R. Storn and K. Price, "Differential Evolution – a simple and efficient Heuristic for global optimization over continuous spaces", Journal Global Optimization. 11, pp. 341 – 359, 1997.

[14] Tang, K., Yao, X., Suganthan, P. N., MacNish, C., Chen, Y. P., Chen, C. M., Yang, Z., "Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization", IEEE Congress on Evolutionary Computation, 2008.

[15] Deep, K. and Thakur, M , "A new crossover operator for real coded genetic algorithms", *Applied Mathematics and Computation*, Vol. 188, No. 1, pp.895–911, 2007.